





UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Sistema de Guiado para Peatones, Ciclistas y Motoristas con Interacción Implícita

Carlos Ramos Mellado

Febrero, 2015



Sistema de Guiado para Peatones, Ciclistas y Motoristas con Interacción Implícita







UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de Información

TECNOLOGÍA ESPECÍFICA DE
TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO FIN DE GRADO

Sistema de Guiado para Peatones, Ciclistas y Motoristas con Interacción Implícita

Autor: Carlos Ramos Mellado

Director: Dr. David Villa Alises

Febrero, 2015

Carlos Ramos Mellado

Ciudad Real – Spain

E-mail universidad: Carlos.Ramos1@alu.uclm.es
E-mail personal: CarlosRamosCriptana@gmail.com

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	**VOCAL**	**SECRETARIO**
Fdo.:	Fdo.:	Fdo.:

Resumen

No hace demasiado tiempo, resultaba increíble llevar en el bolsillo una herramienta que permitiera conocer la localización geográfica exacta del usuario. Mucho menos creíble resultaba imaginar que esa herramienta otorgaría la capacidad de estar siempre conectado a Internet. No obstante, hoy en día esa herramienta existe y se llama *smartphone*.

Con la reciente popularización de los *smartphones* han surgido una larga serie de complementos *wearables* o «llevables» que se comunican con los *smartphones* y proporcionan una nueva forma de interacción con el dispositivo.

En el presente proyecto se pone de manifiesto uno de los problemas más importantes relacionados con el uso de los *smartphones* para la navegación por satélite: la facilidad de, cuando se conduce un vehículo, ocasionar una distracción y en numerosas ocasiones un accidente. Para solucionarlo se plantea una nueva forma de interacción en la que el sistema se encargue de avisar al usuario en el momento que tenga que realizar alguna acción por medio de *wearables*. De este modo se evita que el usuario tenga que mirar la pantalla y se previenen ese tipo de accidentes.

La aplicación desarrollada a lo largo del proyecto se llama *Naviganto* y ofrece al usuario la posibilidad de navegar por satélite guiado por las vibraciones de dos dispositivos: uno colocado en la parte izquierda del cuerpo y el otro situado en la parte derecha. Además, *Naviganto* proporciona al usuario las interacciones comunes de las aplicaciones de navegación como la pantalla o el sonido.

Abstract

Not too long ago it was incredible to carry a tool in your pocket that would allow us to know our exact location. It was much less credible to imagine that this tool would give us the ability to be always connected to the Internet. However, today that tool exists and it is the *smartphone*.

With the recent popularity of smartphones have emerged a long series of *wearables* accessories that communicate with *smartphones* and provide us a new form of interaction with the device.

In this project it is shown one of the most important problems related to the use of *smartphones* for satellite navigation: ease of cause a distraction, and many times an accident. To solve this it is proposed a new form of interaction in which the system is responsible for warning the user, when he has to perform some action through *wearables*. In this way we avoid the user to have to look at the screen and will prevent this type of accident.

The application developed throughout the project is called *Naviganto* and it offers the user the possibility to surf via satellite guided by the vibrations of two devices: one placed on the left side of the body, and the other one located on its right side. Moreover, *Naviganto* provides the user with common interactions of navigation applications such as the screen or the sound.

Agradecimientos

Mis años como estudiante universitario han sido, sin lugar a dudas, los años más intensos de mi vida. Me han ayudado a conocerme mejor y a superar mis propios límites dejando atrás momentos buenos y momentos no tan buenos. Por ello, ahora que me dispongo a comenzar una nueva etapa en mi vida, me gustaría aprovechar este breve espacio para dar las gracias a todas las personas que de alguna forma han hecho posible la finalización de estos estudios.

En primer lugar, y no podría ser de otro modo, me gustaría dar las gracias a mis padres por todo su apoyo, cariño y paciencia. Gracias por creer en mi a lo largo de estos años y permitirme estudiar la carrera que yo elegí. También quiero darle la gracias a mi hermano por servirme como ejemplo y al resto de familiares que de un modo u otro me han ofrecido su ayuda a lo largo de este tiempo.

En segundo lugar, debo dar las gracias todos los compañeros de clase, compañeros de piso y amigos en general que han estado conmigo estos años. Gracias por compartir todas esas alegrías, preocupaciones y pasiones diarias. Vosotros hicisteis el camino más divertido.

Por último, agradezco a David Villa el tiempo invertido en este proyecto y los nuevos conocimientos que me ha permitido adquirir con su dedicación.

Carlos Ramos

A mis padres.

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Listado de acrónimos

TFG	Trabajo Fin de Grado
SGNS	Sistema Global de Navegación por Satélite
GNSS	Global Navigation Satellite System
PDA	Personal Digital Assistant
NHTSA	National Highway Traffic Safety Administration
SO	Sistemas Operativos
API	Application Programming Interface
NAVSTAR-GPS	NAVigation System and Ranging - Global Position System
GPS	Global Position System
GLONASS	Global'naya Navigatsionnaya Sputnikovaya Sistema
BNTS	BeiDou/Compass Navigation Test System
QZSS	Quasi-Zenith Satellite System
IRNSS	Indian Regional Navigation Satellite System
CES	Consumer Electronics Show
HFP	Hands-Free Profile
HID	Human Interface Device Profile
PAN	Personal Area Networking Profile
OSM	Open Street Map
SDK	Software Development Kit
REST	Representational State Transfer
IPO	Interacción Persona Ordenador
GUI	Graphical User Interface
MAC	Media Access Control
COCOMO	COnstructive COst MOdel

Capítulo 1

Introducción

El guiado de personas es una actividad que se lleva desarrollando desde siempre con la ayuda de las estrellas, mapas y brújulas. Afortunadamente, desde que se desarrolló *Transit* en los años sesenta considerado el primer Sistema Global de Navegación por Satélite (SGNS) o Global Navigation Satellite System (GNSS), resulta bastante más sencillo conocer la ubicación de una persona de forma precisa y seguir un determinado camino.

A pesar de que *Transit* se desarrollase en los años sesenta, no fue hasta 1983 cuando *Honda* se planteó utilizar la tecnología de los SGNS para guiar a civiles y desarrolló su sistema de navegación, culminándolo en 1990 para el *Acura Legend*. Desde entonces se democratizó el uso de los SGNS hasta el punto de encontrar en el mercado un elevado número de Personal Digital Assistant (PDA) equipados con sistemas de navegación a finales de la primera década del 2000.

Con la reciente popularización de los *smartphones* se ha extendido aún más el uso de la navegación vía satélite y es común encontrar personas utilizándola por medio de aplicaciones como *Google Maps*, *iOS Maps* o *Nokia HERE*. Como se verá posteriormente, estas aplicaciones disponen de un gran número de opciones y resultan muy útiles permitiendo llegar a cualquier sitio sin necesidad de conocer el camino previamente, pero implican un requisito importante: es necesario ver la pantalla u oír las instrucciones para llevarlas a cabo. Eso no es un gran inconveniente en un coche, pero para los peatones, ciclistas y motoristas el hecho de mirar a la pantalla o intentar oír el smartphone supone una distracción que potencialmente puede provocar un accidente [Val12] y, por tanto, para hacer uso de la navegación vía satélite necesitan otro tipo de interacción con el dispositivo.

Las distracciones del conductor son una de las principales causas de accidentabilidad a nivel mundial. La National Highway Traffic Safety Administration (NHTSA) señaló en 2003 que la distracción de los conductores es la causa de 1,5 millones de accidentes producidos anualmente en todo el planeta [dFCdC03]. Y, según un estudio de la aseguradora *Allianz* de 2014 [Pul14], el 26 % de los accidentes producidos por distracciones se deben a mirar la pantalla del navegador. Esto hace un total de 390,000 accidentes que se podrían evitar anualmente cambiando la interacción con el sistema.

Las autoridades de todo el mundo conocen la relación entre distracciones y accidentes e intentan evitar que se produzcan por medio de su legislación. A día de hoy, en España, el uso de dispositivos como navegadores, cascos y auriculares por parte del conductor está considerado como una infracción grave y acarrea una multa de 200 euros y una pérdida de 3 puntos en el carné de conducir [Ser14].

Para hacer frente al problema de las distracciones a la hora de utilizar los sistemas de navegación, en este trabajo se estudiarán las formas de interacción con el sistema que no supongan una distracción del usuario. Además se hará un repaso por las diferentes tecnologías, plataformas y dispositivos sobre los que se puede cimentar un nuevo sistema para conocer las limitaciones técnicas a las que hay que enfrentarse. Puesto que se trata de un sistema que cambia de estado continuamente las limitaciones que resultarán significativas están relacionadas con la rapidez y la exactitud de las tecnologías y dispositivos.

Finalmente, se desarrollará un sistema de guiado en el que el usuario obtendrá realimentación sin necesidad de ver la pantalla, es decir, por medio de alguna *interacción implícita*. Para ello, el sistema avisará al usuario en el momento que tenga que realizar cualquiera de las posibles acciones. La *interacción implícita* que se pretende conseguir con este trabajo se llevará a cabo por dos medios:

- El **sonido**. Como todos los navegadores actuales nos proporcionará las instrucciones de manera auditiva pero como el trabajo va dirigido a peatones, ciclistas y motoristas y resulta complicado escuchar el dispositivo en un ambiente abierto, es necesaria otra interacción adicional.
- La **vibración**. Necesaria para las situaciones en las que resulte complicado oír el dispositivo y porque está prohibido el uso de cascos o auriculares.

Dado el elevado número de posibles instrucciones que el sistema puede comunicar al usuario por medio de la vibración como giros, cambios de sentido, salidas en una rotonda, etc; será necesario utilizar más de un dispositivo vibratorio. De esta forma, si se coloca un dispositivo en la parte izquierda del cuerpo y otro en la parte derecha se pueden codificar de manera inequívoca los giros a la derecha y a la izquierda haciendo más intuitivas las vibraciones. Esto implica la necesidad de desarrollar un sistema de comunicaciones entre dispositivos pero se consiguen resultados más interesantes.

El proyecto pretende utilizar un teléfono (*smartphone*) y su vibrador junto con un periférico como un reloj (*smartwatch*) o pulsera inteligente (*smartband*). De este modo, cuando haya que girar a la izquierda vibrará uno, cuando haya que girar a la derecha vibrará el otro y cuando haya que dar media vuelta vibrarán ambos. De esta manera, si es colocado el *smartwatch* o *smartband* en la mano izquierda y el teléfono en el bolsillo derecho, cuando haya que realizar un giro a la izquierda solamente vibrará el complemento dejando evidente la acción a realizar (ver Figura 1.1).

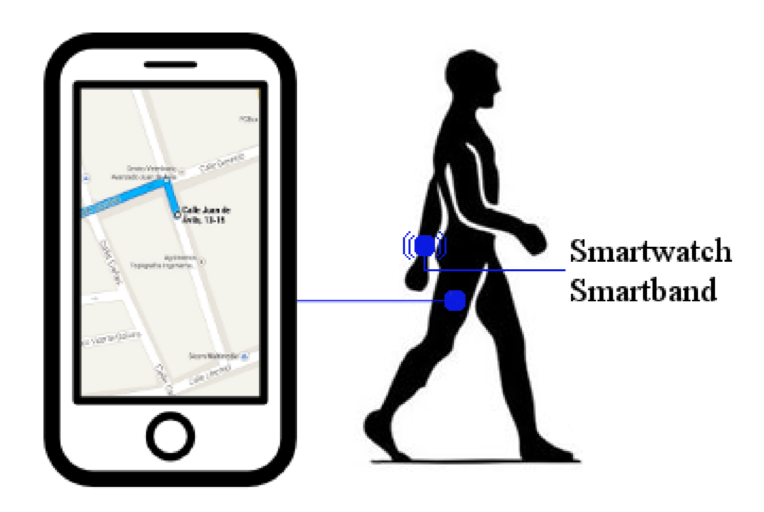


Figura 1.1: Ejemplo del sistema indicando un giro a la izquierda

1.1 Estructura del documento

A continuación se detalla la estructura de este documento para facilitar la lectura del mismo y se describe brevemente el contenido de cada uno de los capítulos que lo componen.

Capítulo 2: Objetivos

El principal objetivo de este proyecto es el diseño e implementación de un sistema de navegación por satélite que se comunique con el usuario por medio de la vibración de alguno de sus componentes.

En este capítulo se detallan los objetivos específicos que se pretenden alcanzar a lo largo del desarrollo del proyecto y las limitaciones asociadas.

Capítulo 3: Antecedentes

En este capítulo se hace una introducción al concepto de *interacción* y sus paradigmas. También se hace un repaso de los diferentes sistemas de navegación vía satélite y las aplicaciones para *smartphone*. Se pondrá especial interés en la forma que tienen para comunicarse con sus usuarios.

Además, se realizarán estudios que permitan realizar buenas elecciones a lo largo del TFG sobre las plataformas de *smartphones*, los complementos vibratorios y los proveedores de mapas y rutas existentes en la actualidad.

Capítulo 4: Método de trabajo y herramientas

El método seleccionado para el desarrollo del trabajo es el *prototipado evolutivo*.

En este capítulo se describe la metodología seguida, además de especificar las herramientas, tanto *hardware* como *software*, utilizadas para la elaboración del proyecto.

Capítulo 5: Desarrollo del proyecto

En este capítulo se encuentra detallado el trabajo realizado especificando cada una de las iteraciones necesarias para el desarrollo del proyecto.

Capítulo 6: Resultados

En este capítulo se describen los resultados obtenidos tras el desarrollo del sistema y se habla sobre los recursos y costes del mismo.

Capítulo 7: Conclusiones

En este capítulo se detallan las conclusiones obtenidas del desarrollo del proyecto, se incluyen posibles usos alternativos del sistema y se proponen posibles líneas de trabajo futuras a desarrollar.

Capítulo 2

Objetivos

EN el presente capítulo se muestran los objetivos que se marcaron para el desarrollo de este proyecto desde la realización del *anteproyecto* de forma general y específica. De esta manera, se determina el alcance del proyecto y los resultados que se esperan tras su implementación.

2.1 Objetivo general

Incentivado por el gran número de accidentes que se producen continuamente por distracciones a la hora de utilizar los navegadores vía satélite actuales, en este trabajo se desarrollará un sistema de navegación por satélite que haga uso de la *interacción implícita*, es decir, un sistema de navegación que avise al usuario de las acciones a realizar sin necesidad de prestarle atención.

Dado que la única *interacción implícita* que ofrecen los navegadores actuales es el sonido, los ciclistas y motoristas no pueden hacer uso de la navegación por satélite sin incrementar considerablemente el riesgo de accidente. Esto es debido a que mirar la pantalla del navegador provoca el 26 % de los accidentes producidos por distracciones [Pul14], resulta complicado oír el dispositivo en ambientes abiertos y el uso de cascos o auriculares esta sancionado en España [Ser14].

Con la finalidad de hacer más segura la navegación para los ciclistas y motoristas, y para facilitar su uso para los peatones, se necesita otro tipo de *interacción implícita*. Así pues, si se descarta la interacción por medio de la vista por no ser implícita y el oído por lo citado anteriormente, sólo quedan disponibles el gusto, el olfato y el tacto. Y, puesto que la interacción con el usuario por medio del gusto y el olfato está muy poco desarrollada, sólo queda el tacto. Por lo tanto, se hará uso del tacto por medio de la vibración de los dispositivos.

En resumen, el objetivo del Trabajo Fin de Grado (TFG) es el «desarrollo de un sistema de navegación por satélite que se comunique con el usuario por medio de la vibración de alguno de sus componentes»

2.2 Objetivos específicos

A partir del objetivo principal descrito en la sección anterior se determinan los objetivos específicos necesarios para alcanzarlo.

2.2.1 Desarrollo del sistema de posicionamiento

Se implementará un sistema que permita conocer la posición del usuario en el globo y la represente en un mapa. Dicho sistema será el encargado de actualizar la posición del usuario cuando cambie su localización. Para ello será necesario tomar algunas decisiones como:

- Elección de la plataforma de desarrollo
- Elección del proveedor de mapas
- Elección de tecnología de posicionamiento

2.2.2 Desarrollo del sistema de navegación

Se implementará un sistema que permita navegar desde la posición del usuario hasta otra posición determinada por el nombre de un lugar. Para ello se tomarán algunas decisiones de diseño como:

- Elección del proveedor de rutas
- Elección del proveedor de *geocoding*[1]

2.2.3 Desarrollo del sistema de reproducción de audio

Se implementará un sistema de reproducción de audio que permita escuchar las instrucciones necesarias para seguir una ruta. Para ello será necesario seleccionar un proveedor de *text to speech*[2].

2.2.4 Desarrollo del sistema de comunicación con otros dispositivos

Se implementará un sistema de comunicación que permita el paso de mensajes entre los diferentes dispositivos del sistema. Para ello será necesario elegir la tecnología o tecnologías (Wi-Fi, Bluetooth, etc) sobre la que se implementará la arquitectura cliente-servidor.

2.2.5 Desarrollo del sistema vibratorio

Se implementará un sistema vibratorio que, con ambos dispositivos, codifique todas las posibles acciones que el usuario pueda realizar. Para ello, hará uso de diferentes frecuencias de vibración en los dispositivos de modo que la codificación resulte intuitiva.

[1]El geocoding es un proceso mediante el cual se obtienen las coordenadas geográficas de un sitio. Para ello basta con introducir una descripción, una dirección postal o el nombre del lugar.

[2]Un servicio text to speech proporciona el audio de un texto introducido

2.3 Limitaciones

Las limitaciones asociadas al proyecto vendrán dadas por las decisiones tomadas durante el desarrollo antes descritas. Por ejemplo, cuando se seleccione una determinada plataforma de desarrollo para *smartphone*, se tendrán las limitaciones de la plataforma como el lenguaje de desarrollo. De igual forma, cuando se elija el proveedor de mapas, la tecnología de posicionamiento, el proveedor de rutas o el proveedor de *geocoding* se heredará la precisión de los servicios que se utilicen.

Capítulo 3

Antecedentes

En este capitulo se hace una introducción a los aspectos más destacables relacionados con el TFG. Se empieza presentando el concepto de interacción y sus paradigmas. Después se ponen en contexto los sistemas de navegación por satélite y sus actuales aplicaciones para smartphone. Más tarde se hace un estudio por las principales plataformas de desarrollo. Luego se indaga en el concepto de *wearable* y se describen los diferentes tipos de *wearables* en función de la forma de conectarse con el teléfono. Finalmente se hace un repaso por los principales proveedores de servicios relacionados con el sistema.

3.1 Interacción persona ordenador

La Interacción Persona Ordenador (IPO) es la disciplina relacionada con el diseño, implementación y evaluación de sistemas informáticos interactivos para el uso de seres humanos y con el estudio de los fenómenos más importantes con los que están relacionados [TTH92]. Esta disciplina se basa en la interacción con el ordenador y establece una serie de paradigmas de interacción.

Baecker y Buxton en 1987 definieron *interacción* como «todos los intercambios que suceden entre la persona y el ordenador». En el modelo clásico de IPO es la persona la encargada de interactuar con el ordenador y decirle qué hacer, es decir, interacción explícita. Pero hay otra forma de proceder: que sea el ordenador el encargado de interactuar con la persona sin intención explícita o conocimiento del usuario, es decir, interacción implícita.

En este TFG se hará uso de la interacción implícita por medio de la vibración y el sonido para comunicarse con el usuario en los momentos necesarios y decirle qué acción ha de realizar. Algunos proyectos universitarios como [Boe12] y [Mat13] ya han utilizado la vibración para el guiado de personas invidentes y, recientemente, han surgido grandes proyectos como *Nike+ Training* [Foo15] y *Lechal* [Eng14] que hacen uso de la vibración en las zapatillas deportivas para el guiado de personas mientras corren.

3.1.1 Paradigmas de interacción

Los paradigmas de interacción representan los ejemplos o modelos de los que se derivan todos los sistemas de interacción. Es una abstracción de los posibles modelos de interacción organizados en grupos con características similares.

En los siguientes apartados se tratarán los paradigmas interactivos actuales: el ordenador de sobremesa, la realidad virtual, la realidad aumentada y la computación ubicua.

Ordenador de sobremesa

Es el paradigma dominante en la actualidad en el que la interacción se realiza por medio de *interfaces gráficas*. Aunque los *smartphones*, *tablets* y portátiles aportan movilidad utilizan el mismo paradigma de interacción con el usuario.

En la figura 3.1 se muestra un diagrama explicativo de las interacciones entre el usuario, el ordenador y el mundo real.

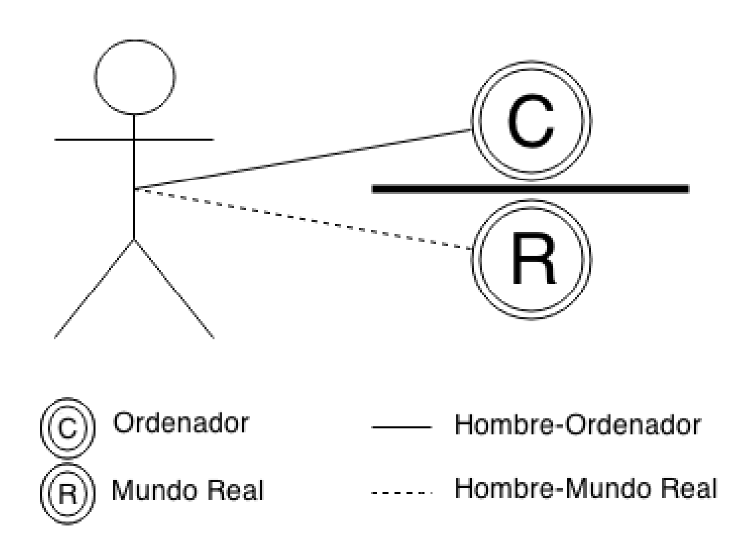


Figura 3.1: Paradigma del ordenador de sobremesa

Computación ubicua

La computación ubicua o computación pervasiva es la evolución del paradigma de computadora de sobremesa en la que el ordenador desaparece desde el punto de vista psicológico. En este paradigma hay que encargarse de la tarea y no de la herramienta que ayuda a realizar esa tarea convirtiendo la computadora en algo invisible, que está por todas partes y que asiste desde el contexto. La computación ubicua es la responsable de las nuevas corrientes de investigación como el *ambiente inteligente* y la *computación diaria*.

En la figura 3.2 se muestra un diagrama explicativo de las interacciones entre el usuario, el ordenador y el mundo real.

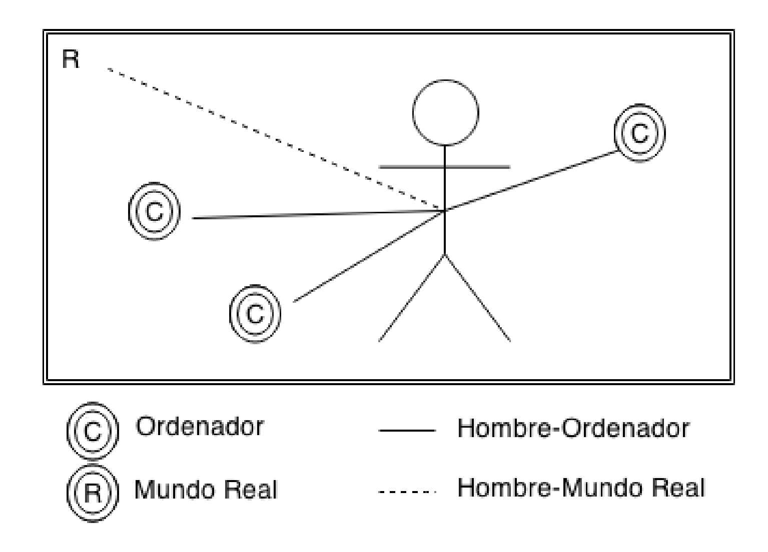


Figura 3.2: Paradigma de la computación ubicua

Los principios sobre los que se apoya la computación ubicua son la descentralización, la diversificación, la conectividad y la simplicidad [Mer13]. Por ello, las herramientas que se desarrollen para el uso de la computación ubicua deben ser invisibles y cercanas al comportamiento del ser humano.

Entre los beneficios de la computación ubicua se pueden destacar la simplicidad o invisibilidad de la interacción y la adaptación al usuario y sus necesidades. No obstante, el uso de la computación ubicua conlleva algunos inconvenientes como la pérdida de privacidad o la escasa cantidad de tareas que se pueden desempeñar.

Este proyecto pertenece al ámbito de la computación ubicua ya que, una vez seleccionadas las opciones en las *interfaces gráficas*, el resto de interacciones que se llevarán a cabo serán transparentes al usuario desapareciendo la computadora y centrándose en el guiado.

Realidad virtual

La realidad virtual, también llamada por algunos investigadores como *ciberespacio*, *realidad artificial*, *ambientes sintéticos* o *ambientes virtuales*; es una simulación por computador, tridimensional e interactiva en la que el usuario se siente introducido en un ambiente artificial. El usuario percibe dicho ambiente como real provocando la percepción de inmersión dentro del mucho virtual.

Entre las ventajas del uso de la realidad virtual se pueden destacar:

- Lo usuarios pueden ver y manipulan objetos en tres dimensiones.
- Ofrece una visión del mundo en primera persona.
- Permite familiarizarse con realidades antes de que sean construidas.
- No ocupa espacio físico.

De igual forma, la realidad virtual conlleva algunos problemas como:

- Alto coste de desarrollo, tanto de tiempo como de dinero.
- Desarrollos muy específicos que no permiten mucha reutilización.
- Gran inversión inicial en equipos.

En la figura 3.3 se muestra un diagrama explicativo de las interacciones entre el usuario, el ordenador y el mundo real.

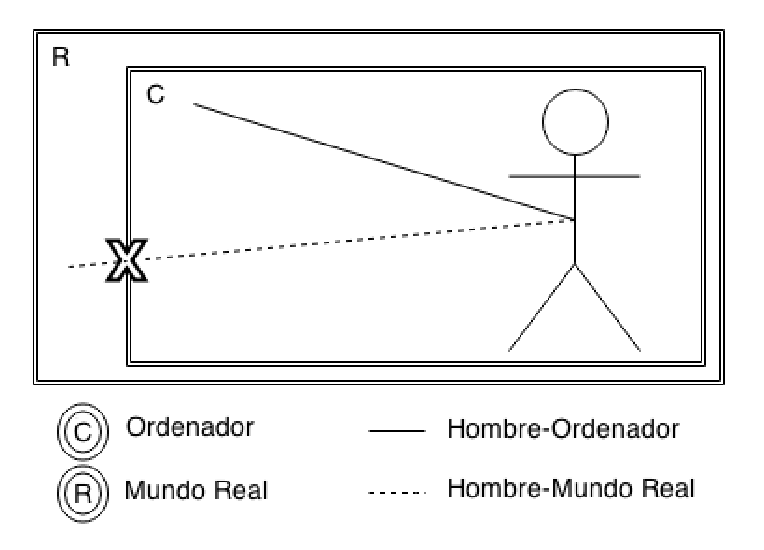


Figura 3.3: Paradigma de la realidad virtual

Realidad aumentada

La realidad aumentada consiste en representar un entorno real sobre el que sólo se añaden algunos objetos artificiales. Puesto que estos objetos dependen del lugar donde se encuentre el usuario y sólo se sobreimprimen dichos objetos sobre la realidad, la realidad aumentada provoca una menor inmersividad que la realidad virtual.

Entre las ventajas del uso de la realidad aumentada se pueden destacar:

- Los usuarios pueden recibir información complementaria a la realidad.
- Puede ayudar a comprender mejor la realidad.

De igual forma, la realidad aumentada conlleva algunos problemas como:

- Necesidad de crear objetos virtuales muy precisos.
- Necesidad de sincronizar temporal y espacialmente la información real y virtual.

En la figura 3.4 se muestra un diagrama explicativo de las interacciones entre el usuario, el ordenador y el mundo real.

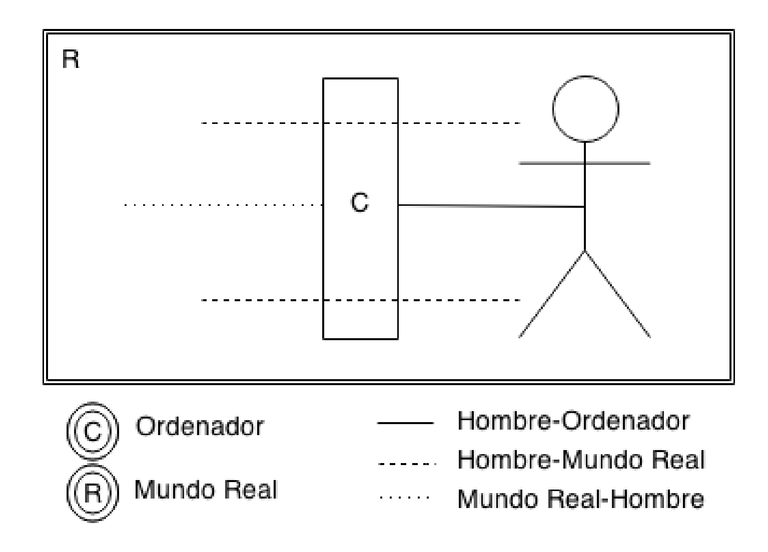


Figura 3.4: Paradigma de la realidad aumentada

3.2 Sistemas de navegación por satélite

Un SGNS o GNSS consiste en una constelación de satélites que transmite señales que permiten determinar las *coordenadas geográficas*, la *altitud* y la *hora* (cuatro dimensiones) de un usuario con mucha exactitud, en cualquier parte del mundo, las 24 horas y en todas las condiciones climatológicas.

El origen de la *navegación por satélite* se remonta a los años sesenta cuando el ejército de Estados Unidos desarrolló *Transit* basado en el *Efecto Doppler* [Wik14a]. Los satélites viajaban por caminos conocidos y transmitiendo una frecuencia conocida. La frecuencia recibida difería ligeramente de la frecuencia de emisión debido al movimiento del satélite. Mediante dicha variación, el receptor puede determinar su ubicación a un lado o al otro del satélite, y varias de estas medidas combinadas con un conocimiento preciso de la órbita del satélite pueden determinar una posición el particular. Este sistema requería que el receptor estuviera casi estático unos 40 minutos para establecer su posición.

La primera vez que apareció el concepto de SGNS para guiar a civiles fue en 1983 cuando *Honda* se planteó utilizar esta tecnología y desarrolló su sistema de navegación para coches. Pero no lo culminó hasta 1990 cuando lo integró en el *Acura Legend* [Par13].

El funcionamiento de los SGNS modernos es más directo. El satélite emite una señal que contiene los datos orbitales necesarios para calcular su posición y el tiempo preciso en el que se transmitió la señal. De este modo, el receptor compara el momento de la emisión con el tiempo de recepción y puede establecer su posición con respecto al satélite. Y, al utilizar varios satélites es posible establecer la posición el globo mediante la trilateración (ver figura 3.5).

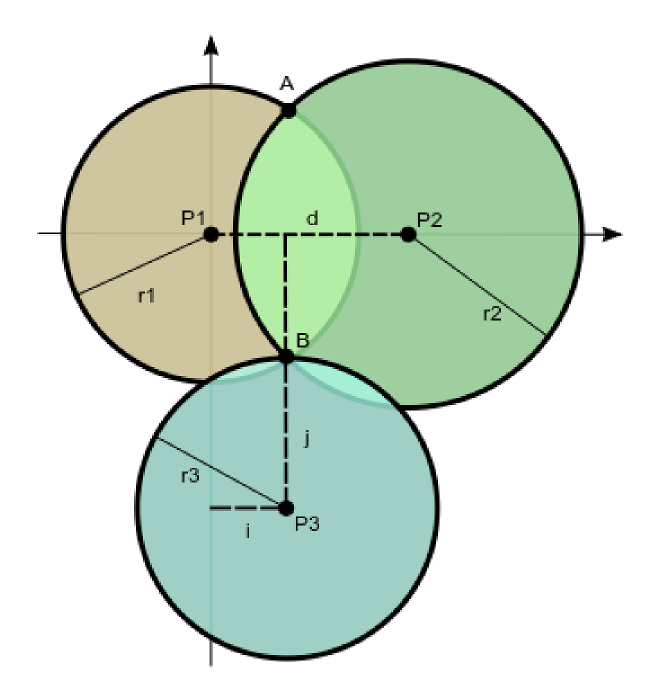


Figura 3.5: Ejemplo de trilateración

Estando en B, se quiere conocer la posición relativa a los puntos de referencia P1, P2, y P3 en un plano bidimensional. Al medir r1 se reduce la posición a una circunferencia. A continuación, midiendo r2, se reduce a dos puntos, A y B. Una tercera medida, r3, devuelve las coordenadas de B. Una cuarta medida también puede hacerse para reducir y estimar el error.

3.2.1 Coordenadas geográficas

Las coordenadas geográficas son el sistema que permite representar una posición en el globo. Este sistema de referencia hace uso de dos coordenadas angulares *latitud* (Norte y Sur) y *longitud* (Este y Oeste) medidas desde el centro de la tierra y expresadas por regla general en grados sexagesimales.

- La **latitud** es el ángulo que forma el Ecuador con el punto que medimos.
- La **longitud** es el ángulo se forma entre el meridiano de Greenwich con el punto que medimos.

3.2.2 Tecnologías actuales

En la actualidad predominan dos tecnologías: Global Position System (GPS) y Global'naya Navigatsionnaya Sputnikovaya Sistema (GLONASS). En los siguientes apartados se resumen sus características más importantes.

No se han incluido en el estudio ninguno de los sistemas que a fecha de hoy no se consideran operativos [Wik14b] como:

- Galileo desarrollado por la Unión Europea.
- BeiDou/Compass Navigation Test System (BNTS) desarrollado por la República Popular de China.
- Quasi-Zenith Satellite System (QZSS) desarrollado por Japón.
- Indian Regional Navigation Satellite System (IRNSS) desarrollado por India.

NAVigation System and Ranging - Global Position System (NAVSTAR-GPS)

Este sistema, también conocido simplemente por GPS es el sistema de navegación desarrollado por los Estados Unidos. Los primeros proyectos relacionados empezaron a desarrollarse en 1973 y en ellos se establecieron las bases de lo que hoy conocemos como GPS. Los primeros satélites se lanzaron en 1978 y el programa fue plenamente operativo en 1995, aunque con un sistema de degradación de señal para los usos civiles que limitaba la señal a una precisión aleatoria entre 15 y 100 metros. En mayo del año 2000 se eliminó dicha limitación permitiendo determinar la posición y la hora de forma precisa, con un error nominal de unos 15 metros.

En la actualidad, esta tecnología es accesible por la mayoría de los *smartphones*. Está formada por una constelación de 24 a 27 satélites que se mueven en órbita de unos 20.000 km, por seis planos y con una inclinación de 55 grados.

GLONASS

Al mismo tiempo que se desarrollaba el GPS, en Rusia también desarrollaron su sistema de navegación GLONASS. Los primeros satélites fueron lanzados en 1982 pero el programa no fue operativo hasta 1996 con unos 9 a 11 satélites y un sistema de degradación de señal para usos civiles parecido al de GPS. Esta limitación hacía que la precisión fuese de unos 30 metros. En 2007 se eliminaron las limitaciones para uso civil y en 2012 se convirtió en un sistema de navegación plenamente funcional cuando se pusieron en órbita más satélites. El sistema GLONASS está optimizado para ser usado en el hemisferio norte y proporciona la posición con un error entre 5 y 15 metros.

En la actualidad, esta tecnología puede ser usada sólo por algunos *smartphones*. Está formada por una constelación de 31 satélites que orbitan a una altitud de 19.100 km, situados en tres planos y con una inclinación de 64,8 grados.

3.2.3 Aplicaciones de navegación para smartphone

Aunque existen una gran cantidad de aplicaciones de navegación para *smartphone* disponibles en las tiendas oficiales de aplicaciones, en este apartado analizaremos solamente las aplicaciones más usadas.

Google maps

Google Maps es una aplicación de navegación lanzada por Google en septiembre de

2008 que se puede utilizar tanto en Android como en iOS, Windows Mobile, Symbian o BlackBerry. Además, Google Maps está disponible en múltiples idiomas.

Entre las características de la aplicación de Google se pueden destacar las siguientes:

- Visualización de la posición en el mapa mostrando el posible error cometido en la *trilateración*.
- Visualización de mapas offline previamente descargados.
- Visualización de la cantidad de tráfico de las carreteras.
- Visualización de edificios en 3D en algunas ciudades.
- «Street View» o posibilidad de ver fotografías a pie de calle.
- Rotación automática del mapa en función de la orientación del dispositivo.
- Cálculo de rutas para peatones, transporte público y vehículos motorizados.

Las interacciones que utiliza Google Maps para comunicarse con el usuario se realizan por medio de la pantalla, el sonido y la vibración. La pantalla muestra continuamente las próximas acciones a realizar y, cuando se estima oportuno, el smartphone vibra y se escucha por los altavoces la instrucción a realizar. La vibración que se utiliza es la misma para todos los tipos de instrucciones.

iOS maps

iOS Maps es la aplicación de Apple lanzada en Septiembre de 2012 para sus dispositivos iPhone, iPad y iPod Touch. Esta aplicación es multilenguaje. Entre sus características se pueden destacar:

- Visualización de la posición en el mapa mostrando el posible error cometido en la *trilateración*.
- Visualización de la cantidad de tráfico de las carreteras.
- Visualización de edificios en 3D en algunas ciudades.
- Rotación automática del mapa en función de la orientación del dispositivo.
- Cálculo de rutas para peatones y vehículos motorizados.

Con respecto a la interacción con el usuario, iOS maps utiliza la pantalla y el sonido. La pantalla mostrando continuamente las instrucciones a seguir y el sonido a la hora de realizar las instrucciones.

Nokia HERE

Nokia HERE, antes conocido como Ovi Maps (2007-2011) y como Nokia Maps (2011-2012), es la aplicación de navegación desarrollada por Nokia. Está disponible para Windows Phone, Symbian y Android; y se encuentra disponible en múltiples lenguajes. Entre sus características es importante destacar:

- Visualización de la posición en el mapa mostrando el posible error cometido en la *trilateración*.

- Visualización de mapas offline previamente descargados.
- Visualización de la cantidad e tráfico de las carreteras.
- Visualización de edificios en 3D en algunas ciudades.
- Rotación automática del mapa en función de la orientación del dispositivo.
- Cálculo de rutas para peatones, transporte público y vehículos motorizados.

Con respecto a la interacción con el usuario, HERE utiliza la pantalla y el sonido. La pantalla mostrando continuamente las instrucciones a desarrollar y, a la hora de realizar las instrucciones, se reproducen sonoramente.

OsmAnd

OsmAnd es la aplicación de navegación de código abierto (GPL v3) para Android basada en los mapas de Open Street Map (OSM). Existe una versión gratuita y otra de pago que permite desbloquear el limite de descargas de mapas offline y ayudar al desarrollo de la aplicación.

Entre las características más importantes se pueden destacar:

- Visualización de la posición en el mapa mostrando el posible error cometido en la *trilateración*.
- Visualización de mapas offline previamente descargados.
- Rotación automática del mapa en función de la orientación del dispositivo.
- Cálculo de rutas para peatones y vehículos motorizados.

La interacción con el usuario se realiza visualizando la pantalla (que puede configurarse para que se encienda antes de cada acción) y por medio del sonido.

3.3 Plataformas smartphone

Dese la aparición de la industria dedicada a la telefonía móvil allá por 1983 cuando la compañía *Motorola* estrenó el primer móvil de la historia, el *Motorola DynaTAC 8000X*, se han sucedido una serie de mejoras continuas hasta ofrecer al usuario la posibilidad de estar continuamente conectado para llegar a lo que hoy conocemos como *smartphone*.

El éxito de los teléfonos inteligentes o *smartphones* radica en que ofrecen la posibilidad de llevar continuamente con el usuario un ordenador en el que se pueden instalar multitud de aplicaciones. Estos programas tienen finalidades muy dispares, desde aplicaciones relacionadas con el ámbito laboral como gestores de correo electrónico y editores de texto, hasta aplicaciones de ocio como juegos y redes sociales.

A lo largo de estos años, los fabricantes se han basado en diferentes Sistemas Operativos (SO) especialmente diseñados para la telefonía móvil a la hora de gestionar los terminales. Estos SO están enfocados en exprimir las capacidades multimedia e inalámbricas de los dispositivos.

Puesto que cada sistema cuenta con sus propias aplicaciones, los SO han ido adquiriendo mayor importancia de forma que existe una guerra por hacerse por la mayor cuota de mercado.

Este TFG pretende aprovechar la popularidad de los *smartphone* para hacer llegar la aplicación desarrollada al mayor número de usuarios. Para ello, habrá que decidir cuál será la plataforma sobre la que se desarrollará el proyecto: *Android*, *iOs*, *Windows Phone* o *Blackberry*.

A pesar de que *Symbian* de Nokia tenía bastante popularidad hace no demasiados años, queda completamente descartado del estudio porque en la actualidad posee menos del 0.5 % de cuota de mercado y está previsto dejar de implantarlo en nuevos terminales a partir de 2016 [Lit13].

3.3.1 Android

Andorid es un SO basado en Linux desarrollado por la *Open Handset Alliance* liderada por Google. Su lanzamiento tuvo lugar en Octubre de 2008 y, desde entonces, se ha hecho con el 81 % [RL13] de la cuota de mercado de los smartphones. La última versión de este SO es la 5.0.1 o «Lollipop» y el lenguaje de programación de aplicaciones es *Java*.

Uno de las principales causas de su éxito es ser un sistema abierto. Esto ha permitido que salgan al mercado una gran multitud de terminales con características y precios muy distintos que han conseguido que llegue a la mayoría de los usuarios de *smartphone*.

Ventajas

Android cuenta con la mayor cuota de mercado y facilidad a la hora de empezar a desarrollar y posteriormente probar la aplicación. Además, para distribuir la aplicación por medio de la tienda oficial, la *Play Store*, sólo hay que pagar previamente 25$.

Por otro lado, gracias a la variedad de fabricantes, Android posee una gran variedad de dispositivos con diferentes características y se puede seleccionar el dispositivo con la potencia y características que se estimen oportunas.

Desventajas

Las principales ventajas también suponen algunos problemas. El hecho de que haya diferentes dispositivos con diferente *hardware* complica garantizar el rendimiento óptimo de la aplicación en las diferentes configuraciones de pantalla, memoria y procesador.

3.3.2 iOS

Es el SO desarrollado por Apple para iPhone aunque posteriormente se portó para otros dispositivos de la compañía como iPad o iPod Touch. Su lanzamiento tuvo lugar en Junio de 2007 junto con el primer iPhone y en la actualidad cuenta con un 12.9 % [RL13] de la

cuota de mercado. La última versión de este SO es la 8.1.2 y el lenguaje de programación de aplicaciones es *Objetive-C*.

Apple iOS ha sido durante mucho tiempo la referencia de los desarrolladores de aplicaciones ya que fue el primero en incorporar una tienda de aplicaciones: *AppStore*. Con ello, permitió desarrollar a terceros aplicaciones para sus dispositivos y mantenerlas bajo el control de Apple.

Ventajas

Puesto que el SO está diseñado específicamente para una configuración de *hardware*, iOS permite explotar al máximo sus capacidades y desarrollar aplicaciones para un pequeño grupo de dispositivos bastante potentes.

Desventajas

Apple no permite instalar de forma legal aplicaciones que no hayan sido validadas por medio de su *AppStore*. Si queremos desarrollar aplicaciones hay que pagar una licencia anual de 99$.

Por otro lado, los dispositivos de Apple son en general bastante caros y no están al alcance de todo el mundo.

3.3.3 Windows Phone

Es el SO desarrollado por Microsoft sucesor de Microsoft Mobile y llamado originalmente Pocket PC. Fue presentado en 2010 y en la actualidad cuenta con el 3.7 % [RL13] de la cuota de mercado. La última versión del SO es la 8.1 y soporta los lenguajes programación *C#* y *Visual Basic .NET*.

En 2011 se anunció una alianza con Nokia por la cual se convertirá en el principal SO de la compañía finlandesa. Por ello, a partir de 2016 los teléfonos que antaño utilizaban Symbian utilizarán Windows Phone.

Ventajas

Microsoft provee de un entorno de trabajo como *Visual Studio*, con una gran cantidad de librerías, que permite que la programación resulte lo más sencilla posible.

Desventajas

La mayor desventaja de Windows Phone viene dada por su tardía llegada al mercado que le ha hecho difícil competir con los otros SO. Estos ya tienen bastante cuota de mercado y, al no ofrecer grandes saltos de calidad, no ha podido convencer a los usuarios.

Por otro lado, para publicar aplicaciones en la tienda oficial, es necesario pagar una licencia de 75€.

3.3.4 Blackberry

Blackberry es un SO desarrollado por la empresa canadiense *Research In Motion*. Tuvo sus inicios en 1999 incorporando funciones típicas hoy en día como acceso al correo electrónico e Internet. En la actualidad tiene un 1.7 % [RL13] de cuota de mercado. La última versión de este SO es Blackberry 10 y el lenguaje de programación de las aplicaciones es *Java*.

Aunque este sistema está orientado especialmente a empresas y profesionales, tuvo un gran éxito entre los jóvenes gracias a su servicio de mensajería instantánea.

Ventajas

No es necesario pagar para subir aplicaciones a su tienda oficial de aplicaciones llamada *App World*. Sólo es necesario registrarse como desarrollador y que aprueben la aplicación creada.

Para determinados tipos de usuario puede considerarse una ventaja que la mayoría de terminales dispongan de un teclado físico.

Desventajas

De igual modo, disponer de un teclado físico limita las posibilidades de desarrollo frente a dispositivos con pantallas táctiles.

Además, su tienda de aplicaciones cuenta con poco apoyo por parte de los desarrolladores y, en consecuencia, dispone de muchas menos aplicaciones que la competencia.

3.4 Wearables

La palabra *wearable* hace referencia al conjunto de aparatos y dispositivos que se sitúan en alguna parte del cuerpo interactuando continuamente con el usuario y con otros dispositivos para realizar alguna función específica. Hoy en día es posible encontrar relojes, pulseras, colgantes, anillos, gafas, ropa o zapatillas que cumplen con esta descripción.

El término *wearable* tiene raíz inglesa y se puede traducir como «llevable» o «vestible» haciendo referencia a computadoras corporales o «llevables» por el usuario. Bajo esta visión, el ordenador deja de ser un elemento externo al usuario que usa en determinados espacios para convertirse en un elemento con el que interactúa continuamente y lo lleva a todos sitios.

Dentro de esta definición no se considera *wearable* a otros elementos de uso diario como la televisión, la cafetera o el lector de ebooks pese a llevar procesadores. Esto es así porque no forman parte de las personas, es decir, no son «llevables» o «vestibles».

Se estima que los orígenes de la tecnología *wearable* data de la década de 1970 pero no ha sido hasta la década del 2010 cuando la tecnología ha evolucionado lo suficiente para atraer a los consumidores.

Las grandes compañías están apostando por la tecnologías *wearables*. En la Consumer Electronics Show (CES)[1] de enero de 2014, grandes empresas como Intel, Adidas, Sony o Rebook han presentado diferentes complementos *wearables*. Más tarde, en abril, Google presentó sus *Google Glass* y se espera la llegada del *AppleWatch* de Apple para la primavera de 2015. Todo ello hace pensar que los wearables han llegado para quedarse y que los próximos años sufrirán una gran expansión.

En este proyecto se pretende aprovechar la creciente popularidad de los *wearables* utilizando alguno del mercado como complemento vibratorio. De esta forma se conseguirá que el sistema final pueda ser usado por la mayor cantidad de gente posible.

3.4.1 Wearables vibratorios para smartphone

A pesar de la gran variedad de dispositivos *wearables*, no todos tienen la misma forma de interacción con el usuario y para este TFG sólo es interesante destacar los que se puedan sincronizar con un *smartphone* y sean vibratorios.

A continuación se habla de los diferentes tipos de wearables que disponen de vibración separándolos por la forma que tienen de conectarse con el *smartphone*.

Perfiles bluetooth

La mayoría de dispositivos de bajo coste como colgantes, anillos o pulseras utilizan este método de conexión con el *smartphone*. Básicamente se sincronizan con el móvil con el perfil de manos libres o Hands-Free Profile (HFP) para que cuando se reciba una llamada el *wearable* vibre. Como consecuencia de ello, resulta imposible desarrollar aplicaciones para estos complementos porque no es posible controlar a voluntad la vibración del *wearable* y, de hacerlo vibrar simulando una llamada, no se podría diferenciar entre una llamada real o una simulada.

Bluetooth con aplicación propietaria

Otros dispositivos como pulseras o relojes se sincronizan con el *smartphone* por medio de bluetooth y sus perfiles Human Interface Device Profile (HID) o Personal Area Networking Profile (PAN). Gracias a ello pueden comunicarse con la aplicación propietaria instalada en el *smartphone*, configurar los wearables y recibir la información almacenada en el dispositivo.

Este tipo de conexión es bastante común en los *cuantificadores personales* pero, al tratarse de *software* propietario, es necesaria una API para poder realizar aplicaciones para ellos. Sorprendentemente sólo la empresa Fitbit[2] proporciona una API pero no permiten manipular el vibrador de su dispositivo.

Bluetooth con Google Play services

Este tipo de conexión es la que realizan los *wearables* que utilizan como SO Android

[1] http://www.cesweb.org/

[2] http://dev.fitbit.com/

Wear para comunicarse con el *smartphone* Android. Esto permite que las notificaciones del *smartphone* lleguen al *wearable* de forma transparente al desarrollador y se instalen aplicaciones en el complemento por medio del *smartphone*.

A pesar de que Android Wear fue lanzado por Google en marzo de 2014 se trata de una plataforma de desarrollo muy potente porque incorpora la mayoría de la API de Android [Goo14a]. Gracias a ello, se puede controlar el vibrador del *wearable* a voluntad.

3.5 Proveedores de mapas, rutas y geocoding

Para la realización en este TFG del sistema de guiado por satélite será necesario utilizar alguno de los proveedores de mapas rutas y *geocoding*. A continuación se muestran principales proveedores de estos servicios. Sobra decir que, salvo restricciones de licencia, es posible utilizar los servicios de forma combinada.

Google

Google es el proveedor de mapas, rutas y *geocoding* más conocido de la actualidad. Aunque ofrece los tres servicios como servicios web accesibles por medio de API y están disponibles para todas las plataformas, poseen una gran integración con Android.

Por un lado, la API de mapas permite insertar mapas de todas las partes del mundo con el logotipo de Google en cualquier aplicación. De forma gratuita es posible realizar hasta 2.500 [Goo14b] solicitudes de mapas con 25.000 muestras y con una resolución de 640 x 640.

Por otro, el servicio de rutas deja calcular hasta 2.500 rutas diferentes con hasta 10 hitos por solicitud. Este servicio permite diferenciar entre rutas para peatones, ciclistas, vehículos motorizados o transporte público.

Finalmente, el servicio de geocoding de Google permite realizar gratuitamente hasta 2.500 peticiones al día. Además, se pueden realizar búsquedas por números de casa.

Bing

Es el proveedor de mapas de Microsoft que proporciona una API con interface Representational State Transfer (REST) para integrar los servicios de mapas, *geocoding* y rutas en cualquier aplicación.

Tanto si la aplicación que se desarrolle está disponible para la descarga gratuita como si es de pago, Bing permite un máximo de 125.000 [Mic14] transacciones por año gratuitamente. En estas peticiones van incluidos todos los servicios.

HERE

Es un proveedor de mapas, rutas y *geocoding* de Nokia que dispone de Software Development Kit (SDK) para el desarrollo de aplicaciones en iOS y Android.

HERE permite realizar un total de 100.000 [Nok14] peticiones gratuitas al mes entre imágenes de satélite, *geocoding* y rutas. Es necesario destacar que sólo se pueden calcular rutas para vehículos motorizados y personas, y que la versión de *geocoding* gratuita no permite buscar al nivel de números de casa.

Open Street Map

OSM es un proyecto colaborativo para crear mapas libres y gratuitos. Por ello, es posible utilizar sus mapas y rutas bajo la licencia ODbL [Coa14a].

Su servicio de rutas llamado *MapQuest* permite la opción de calcularlas para peatones, ciclistas y vehículos motorizados. Para ello es necesario registrarse gratuitamente y obtener una clave de usuario.

Por otro lado, el servicio de geocoding de OSM se llama *Nominatim* y funciona sobre servidores mantenidos por medio de donaciones y con una capacidad limitada. Por ello, dicen poner el límite en «un máximo absoluto de 1 petición» [Coa14b] aunque es posible realizar varias. Este servicio no permite realizar búsquedas a nivel de números de casa.

Capítulo 4

Método de trabajo y herramientas

En este capitulo se expone la metodología seguida para llevar a cabo el proyecto y se justifica por qué se considera el método más satisfactorio a la hora de implementar este TFG. Además, se seleccionan y nombran las diferentes herramientas utilizadas para el desarrollo del mismo tanto *software* como *hardware*.

4.1 Metodología de trabajo

Al comenzar este TFG sólo se tenía una idea general del sistema que se quería desarrollar. Puesto que los requisitos disponibles no eran detallados, se decidió optar por un modelo de desarrollo *evolutivo* para ir definiendo las especificaciones a lo largo del progreso del proyecto. En particular, el de *prototipos desechables* o *prototipado evolutivo*.

El desarrollo *evolutivo* se basa en la idea de crear una implementación inicial, exponerla a los comentarios del usuario y refinarla a través de las diferentes versiones hasta conseguir un sistema adecuado [Som05].

En la Figura 4.1 se muestra el diagrama de flujo seguido por un modelo evolutivo en el que se puede ver como las actividades de especificación, desarrollo y validación se realizan de forma concurrente con una rápida retroalimentación entre ellas.

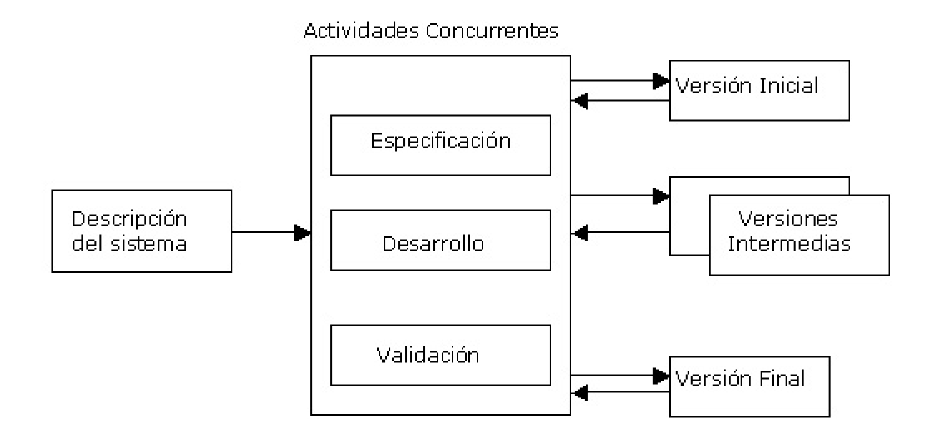


Figura 4.1: Diagrama de flujo del modelo de desarrollo evolutivo

4.1.1 Prototipado evolutivo

El *prototipado evolutivo* es un modelo de desarrollo de software basado en la realización de prototipos funcionales hasta llegar a un producto final. Se parte de los requisitos mejor comprendidos y con mayor prioridad, y se van definiendo los detalles conforme avanza el desarrollo de las diferentes versiones.

Un cliente, a menudo, define un conjunto de objetivos generales para el *software*, pero no identifica los requisitos detallados de entrada, proceso o salida. De igual forma, el responsable del desarrollo del *software* puede no estar seguro de la eficacia de un algoritmo o de la forma en que debería plantearse la IPO. En estas y otras muchas situaciones, un paradigma de construcción de prototipos puede ofrecer el mejor enfoque [Pre10].

El esquema general del *prototipado evolutivo* se puede ver en la Figura 4.2. El paradigma comienza con una recolección de requisitos, es decir, el desarrollador y el cliente definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas donde se necesita más definición. Entonces aparece un diseño rápido centrado en los aspectos que serán visibles para el cliente y se construye un prototipo. Este prototipo es evaluado por el cliente y se utiliza para refinar los requisitos del software. Esta iteración se repite hasta llegar al resultado final.

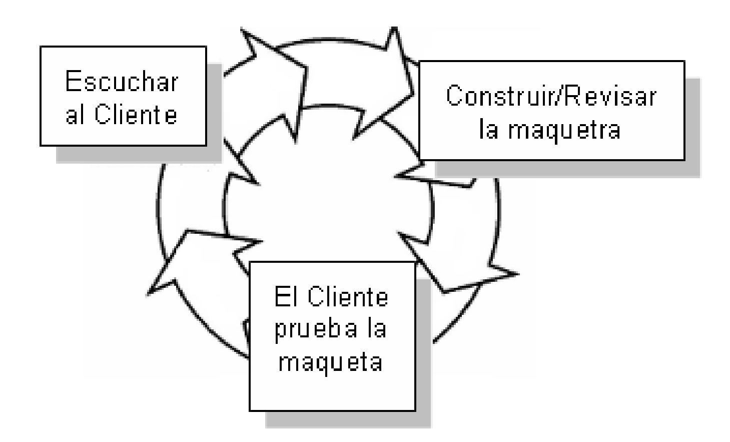


Figura 4.2: Paradigma de construcción de prototipos

Entre las ventajas del uso del *prototipado evolutivo* hay que destacar:

- La continua obtención de prototipos permite al cliente observar y dirigir el desarrollo del *software*.
- Las funcionalidades más importantes son desarrolladas en las primeras fases del proyecto y, por tanto, son las funcionalidades que más veces se ha probado su correcto funcionamiento.
- Como se evalúan prototipos intermedios, es más fácil comprobar si los requisitos planteados son correctos y viables.

Por otro lado, el *prototipado evolutivo* también puede presentar algunas desventajas:

- Resulta difícil estimar el coste final del proyecto debido al desconocimiento del número de iteraciones a realizar.
- Resulta muy difícil y costoso realizar cambios en la arquitectura del software.

Debido a estos problemas, aplicar el *prototipado evolutivo* resulta muy complejo para grandes proyectos pero resulta conveniente utilizarlo en sistemas pequeños y de tamaño medio como el planteado en este TFG. Para proyectos más grandes es recomendable utilizar un modelo híbrido entre el modelo en cascada, para desarrollar las partes bien conocidas, y un enfoque evolutivo para los aspectos que presenten mayor incertidumbre.

4.2 Herramientas

En esta sección se detallan las herramientas que han sido empleadas para la realización de este TFG, desde las necesarias para implementar el proyecto hasta las utilizadas para la realización del presente documento.

4.2.1 Hardware

Entre los medios necesarios para la puesta en marcha del sistema y la realización de las pruebas, se han requerido los siguientes medios *hardware*:

- **Computador** Para el desarrollo de la aplicación y generar la documentación ha sido necesario emplear un ordenador de propósito general. En este caso se ha utilizado un ordenador portátil Asus X54H [1].
- **Smartphones** Para las pruebas han sido necesarios varios *smartphones* con diferentes versiones de Android. En todos se ha testeado tanto la aplicación de navegación como la aplicación para los complementos vibratorios:
  - Nexus 5 [2] con Android 5.0.1.
  - Nexus One [3] con Android 2.3.7.
  - HTC Wildfire S [4] con Android 2.3.5.
- **Smartwatch** Para realizar las pruebas para los complementos vibratorios con Android Wear se ha utilizado el LG G Watch R [5].

4.2.2 Software

A continuación se nombran las diferentes herramientas *software* empleadas distinguiendo entre diferentes categorías:

[1] http://www.asus.com/Notebooks_Ultrabooks/X54H/
[2] https://www.google.es/nexus/5/
[3] https://en.wikipedia.org/wiki/Nexus_One
[4] https://en.wikipedia.org/wiki/HTC_Wildfire_S
[5] http://www.lg.com/es/wearables/lg-LGW110-g-watch-r

Sistemas Operativos

- **Elementary OS** [6] Para la elaboración del trabajo se ha elegido esta distribución GNU/-Linux como SO del computador de desarrollo. Concretamente en su versión 0.3 *Freya*.
- **Android** [7] Para desarrollar la aplicación de navegación y algunos complementos vibratorios se seleccionó el SO Android por ser la plataforma con más cuota de mercado y variedad de dispositivos.
- **Android Wear** [8] Para desarrollar la aplicación de los complementos vibratorios se utilizó Android Wear ya que es la única plataforma de *wearables* que permite acceder al vibrador incorporado.

Herramientas de desarrollo

- **Android Studio** [9] Para la creación de las aplicaciones del sistema de navegación desarrollado se selecciono Android Studio en su versión 1.0.1. ya que se trata de un entorno de desarrollo completamente orientado a Android y Android Wear.
- **Git** [10] Para llevar un control de versiones y así mantener un historial de cambios, tanto en el desarrollo de las aplicaciones como en el desarrollo del proyecto, se ha utilizado Git. El servicio de repositorio empleado ha sido proporcionado por Bitbucket [11].

Librerías

- **Android support** [12] Para lograr la compatibilidad entre las diferentes versiones de Android y de sus API se utilizó esta librería suministrada por Google en su versión 4.21.0.3. También se utilizó **Android support appcompat** en su versión 7.21.0.3 para la compatibilidad del menú lateral.
- **Play services** Para la comunicación con los dispositivos *wearables* se utilizó esta librería. Concretamente en su versión 6.5.87.
- **Commons Lang** [13] Se utilizaron algunos de sus métodos del paquete `java.util`. Se empleó la versión 3.3.2.
- **Osmdroid** [14] Para acceder a los mapas colaborativos de OSM se utilizó esta librería en su versión 4.2.

---

[6] http://elementaryos.org/
[7] https://developer.android.com/index.html
[8] https://developer.android.com/wear/index.html
[9] https://developer.android.com/sdk/index.html
[10] http://git-scm.com/
[11] https://bitbucket.org/
[12] https://developer.android.com/tools/support-library/index.html
[13] https://commons.apache.org/proper/commons-lang/
[14] https://github.com/osmdroid/osmdroid

- **Osmbonuspack** [15] Para realizar algunas acciones sobre los mapas de OSM no disponibles en Osmdroid se utilizó esta librería en su versión 4.9.
- **Gson** [16] Esta librería es la encargada de convertir objetos Java en su representación en JSON y viceversa. Es imprescindible para usar *Osmdroid* y se utilizó la versión 2.3.1.
- **SLF4J** [17] La librería Simple Logging Facade for Java (SLF4J) proporciona una API de registro Java a través de patrón de fachada simple. Es imprescindible para el uso de *Osmdroid* y se utilizó en la versión 1.5.8.

Documentación y gráficos

- **Emacs** [18] El editor de texto «extensible, personalizable, auto-documentado y de tiempo real» ha sido utilizado en su versión 23.4.1 para la elaboración de la documentación del TFG utilizando los modos «latex» y «flyspell».
- **LaTeX** [19] Para la elaboración de este documento se ha empleado esta herramienta de creación de documentos profesionales. Más concretamente, la implementación *TeX Live* [20].
- **BibTex** [21] Se ha empleado esta herramienta destinada a la creación de referencias para documentos escritos en LaTeXpara hacer la bibliografía de este documento.
- **draw.io** [22] Se ha utilizado esta aplicación web para crear la mayoría de diagramas de este documento.
- **Gimp** [23] Se ha usado este editor de imágenes avanzado para crear y retocar algunas imágenes del documento. Concretamente en su versión 2.8.14.

[15] https://code.google.com/p/osmbonuspack/
[16] https://code.google.com/p/google-gson/
[17] http://www.slf4j.org/
[18] https://www.gnu.org/software/emacs/
[19] http://www.latex-project.org/
[20] `https://www.tug.org/texlive/`
[21] http://www.bibtex.org/
[22] https://www.draw.io/
[23] http://www.gimp.org/

Capítulo 5

Desarrollo del proyecto

En este capítulo se describe el proceso de desarrollo del sistema de navegación por satélite. Se empieza enumerando los requisitos originales del sistema y se explica cómo se realizarán las pruebas. Más tarde se explica iteración a iteración las decisiones tomadas, los prototipos desarrollados y las pruebas realizadas sobre ellos.

5.1 Especificación de requisitos

Como se comentó anteriormente (ver sección 4.1) los requisitos del sistema se han ido detallando a lo largo del desarrollo del proyecto ya que, originalmente, sólo tenía una idea muy general del sistema a desarrollar. A continuación se muestran estos requisitos generales sobre los que se partió:

- El sistema debe poder guiar a peatones, ciclistas y motoristas.
- El sistema debe desplegarse sobre alguna plataforma de *smartphone*.
- El sistema debe ser compatible con la mayoría de versiones de la plataforma.
- La interacción con el sistema debe desarrollarse de forma implícita y ser válida para peatones, ciclistas y motoristas.
- Los complementos necesarios para la interacción deben estar disponibles en el mercado y estar al alcance del público en general.
- El sistema debe implementar las características básicas del resto de aplicaciones del mercado como mostrar la posición en el mapa, rotar el mapa en función de nuestra orientación o visualizar la ruta que se va a seguir.

5.2 Pruebas

Para describir las pruebas realizadas sobre los diferentes prototipos en este documento se ha utilizado el patrón «Given-When-Then». Este patrón divide el proceso en 3 etapas:

- **Given** (Dado): Condiciones previas sobre las que se producen los eventos.
- **When** (Cuando): Operaciones específicas que se producen.
- **Then** (Entonces): Resultados esperados.

5.3 Proceso de desarrollo

A continuación se indican las iteraciones realizadas durante el desarrollo del TFG detallando los objetivos, el diseño, la implementación y las pruebas de cada una de ellas.

5.3.1 Iteración 1: Mostrar mapa

En la primera iteración se marcó como objetivo mostrar una posición cualquiera en el mapa por medio de una aplicación de *smartphone*.

Diseño

Para realizar esta primera aproximación hay que tomar tres importantes decisiones de diseño:

- Elegir la plataforma de *smartphone* sobre la que desplegar la aplicación.
- Elegir el proveedor de mapas que proporcionará las imágenes a mostrar.
- Elegir la versión objetivo de la plataforma de desarrollo.

En primer lugar, tras revisar las diferentes plataformas disponibles para *smartphone* (ver sección 3.3) se estipuló que la mejor alternativa es Android. Por un lado, Android posee la mayor cuota de mercado (81 %) y dispone de una gran variedad de dispositivos en una amplia gama de precios. Por otro lado, Android es un SO que se integra fácilmente con la plataforma de complementos Android Wear (ver sección 3.4.1) que es la única que permite manipular el vibrador del *wearable* a voluntad.

En segundo lugar, entre los proveedores de mapas existentes (ver sección 3.5) se seleccionó Open Street Map. Se consideró la mejor elección porque provee de mapas de gran calidad completamente gratuitos. Además, si se encontrase cualquier tipo de error en los mapas suministrados, se podría corregir porque es un proyecto colaborativo. Para simplificar el uso de OSM se utilizó la librería *Osmdroid* tal y como se dijo en la sección 4.2.2.

En tercer y último lugar, se seleccionó como objetivo del desarrollo la API de Android 21, también conocida como Android 5.0 *Lollipop*, por ser la más nueva y poseer el mayor número de funcionalidades. De todos modos, se aseguró la compatibilidad desde la API de Android 10, también llamada Android 2.3 *Gingerbread*, por medio de la librería *Android support* para asegurarse de tener compatibilidad con la mayoría de dispositivos Android. El 99.3 % de los dispositivos del *Google Play Store* según *Android Studio*.

Implementación

Para visualizar los mapas de OSM con la ayuda de la librería *Osmdroid* basta con añadir una vista del tipo `org.osmdroid.views.MapView` al `layout` de nuestra aplicación (ver listado 5.1) y seleccionar el punto del mapa que se desea centrar en la pantalla (ver listado 5.2) por medio de sus coordenadas geográficas.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
        xmlns:tools="http://schemas.android.com/tools"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <org.osmdroid.views.MapView android:id="@+id/map"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent" />

</LinearLayout>
```

Listado 5.1: Ejemplo de `layout` usando `org.osmdroid.views.MapView`

```java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        MapView map = (MapView)findViewById(R.id.map);
        map.setMultiTouchControls(true);
        map.getController().setZoom(18);

        GeoPoint startPoint = new GeoPoint(39.40540171, -3.12204771);
        map.getController().setCenter(startPoint);
    }
}
```

Listado 5.2: Ejemplo de `activity` mostrando un punto de un mapa en específico

Por muy sencillo que pueda parecer, durante la implementación se detectó un problema: el mapa no se centraba en la posición seleccionada. Dejaba dicha posición en la esquina superior izquierda y no en el centro de la pantalla. Estudiando el problema se descubrió que se trataba de un *bug* documentado [1] de *Osmdroid* y que se resolvería en la próxima versión de la librería.

Para paliar dicho problema, y siguiendo con las instrucciones detalladas en el *bug*, se procedió a implementar un método que lo resolviera (ver listado 5.3).

Pruebas

Al finalizar la implementación del primer prototipo se realizó la siguiente prueba:

- Para determinar la correcta carga de los mapas en diferentes localizaciones

[1] https://github.com/osmdroid/osmdroid/issues/22#issuecomment-43092313

```java
private void centerMap(final GeoPoint loc) {
  mMap.getViewTreeObserver().addOnGlobalLayoutListener(
        new ViewTreeObserver.OnGlobalLayoutListener() {
    @Override
    public void onGlobalLayout() {
      mMap.getViewTreeObserver().removeGlobalOnLayoutListener(this);
      mMap.getController().setCenter(loc);
    }
  });
}
```

Listado 5.3: Método utilizado para centrar el mapa en cualquier posición

Dado	Diferentes coordenadas geográficas
	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se ejecuta la aplicación
Entonces	Se muestra dicha ubicación centrada en un mapa

5.3.2 Iteración 2: Mostrar posición actual en el mapa

Una vez evaluado el prototipo de la iteración 1, se procedió a añadir una nueva funcionalidad: mostrar la posición actual en la aplicación.

Diseño

Para poder mostrar la ubicación actual, es necesario primero conocerla. Para ello, se analizaron en la sección 3.2.2 los diferentes sistemas con los que obtener las coordenadas geográficas del dispositivo en la actualidad. Puesto que GLONASS no estuvo operativo para uso civil hasta 2012 no todos los *smartphones* disponen en la actualidad de esta tecnología y deja una única opción razonable: NAVSTAR-GPS.

Implementación

Para hacer uso del GPS del *smartphone* en Android sólo es necesario definir un `LocationListener` e indicar por medio de un `LocationManager` qué clase lo implementa y cada cuánto tiempo se tiene que ejecutar (ver listado 5.4).

```java
if (mLocationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
    mLocationManager.requestLocationUpdates(LocationManager.
        GPS_PROVIDER, 1000, 1, this);
}
```

Listado 5.4: Ejemplo de uso de `LocationManager` utilizando la propia clase como `LocationListener` y 1 segundo de intervalo entre actualizaciones

Para que una clase implemente un `LocationListener` debe contener cuatro funciones:

- **`onLocationChanged`** Para indicar qué hacer cuando nuestra ubicación cambia.
- **`onProviderDisabled`** Para indicar qué hacer cuando se desactiva el GPS.
- **`onProviderEnabled`** Para indicar qué hacer cuando se activa el GPS.
- **`onStatusChanged`** Para indicar qué hacer cuando cambia el estado del proveedor del servicio GPS. Los posibles estados son *fuera de servicio*, *temporalmente no disponible* y *disponible*.

Resulta obvio que sólo se necesita rellenar el método `onLocationChanged` con el código necesario para mostrar una señal en el mapa (ver listado 5.5). Para simplificar la forma de dibujar la posición en el mapa y mostrar el error cometido por la trilateración (ver sección 3.5) de forma visual, se ha escrito una clase llamada `LocationOverlay` en base a la clase contenida en *Osmbonuspack* (ver sección 4.2.2) con el mismo nombre. Lo único verdaderamente significativo que diferencia ambas clases es el icono utilizado para especificar la posición en el mapa.

```
public void onLocationChanged(Location loc) {
    GeoPoint myLocation = new GeoPoint(loc);

    if (!mLocationOverlay.isEnabled()) {
        mLocationOverlay.setEnabled(true);
    }
    mLocationOverlay.setLocation(myLocation);
    mLocationOverlay.setAccuracy((int)loc.getAccuracy());

    centerMap(myLocation);
}
```

Listado 5.5: Ejemplo de implementación de `LocationListener` utilizando para mostrar la localización un `LocationOverlay`

Pruebas

Al finalizar la implementación del segundo prototipo se realizó la siguiente prueba:

- Para determinar la correcta carga de los mapas en diferentes localizaciones

Dado	Diferentes localizaciones geográficas
	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se ejecuta la aplicación
Entonces	Se muestra dicha ubicación centrada en un mapa

5.3.3 Iteración 3: Orientar mapa

Una vez evaluado el prototipo de la iteración 2 se introdujo una nueva funcionalidad: rotar el mapa en función de la orientación del dispositivo. Si por ejemplo se camina por una calle recta en dirección Oeste, en la pantalla se vería una recta vertical. De igual manera, si por ejemplo se camina por una calle en dirección Sur y se gira hacia el Este, en la pantalla se vería un giro a la izquierda.

Diseño

Para poder rotar el mapa en función de la orientación es necesario conocer dónde se encuentran cualquiera de los puntos cardinales: Norte, Sur, Este u Oeste.

Prácticamente todos los *smartphones* actuales disponen de un componente llamado *sensor de orientación* que permite determinar la orientación espacial del teléfono por medio de tres valores expresados en grados:

- **Roll** Mide la inclinación del móvil en ralación a los laterales. Desde -90º con el lateral izquierdo levantado, hasta 90º con el lateral derecho levantado.
- **Pitch** Mide la inclinación del móvil en relación a la parte anterior y posterior. Desde -90º que corresponde a la posición vertical, hasta los 90º que corresponde con la parte anterior del móvil, pasando por 0º cuando el teléfono se encuentra a nivel.
- **Azimuth** Mide el punto cardinal Norte en sentido horario de 0º a 360º

Implementación

Para hacer uso del sensor de orientación del *smartphone* Android basta con registrar el uso del `Sensor` de orientación por medio de un `SensorManager` (ver listado 5.6) indicando que clase implementa los métodos necesarios para manejar el sensor de orientación. En este caso la propia (`this`).

```
mSensorManager =
    (SensorManager)getSystemService(Context.SENSOR_SERVICE);
mOrientation =
    mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
mSensorManager.registerListener(this, mOrientation, SensorManager.
    SENSOR_DELAY_NORMAL);
```

Listado 5.6: Ejemplo de registro de un `Sensor` de orientación con `SensorManager`

El sensor de orientación en Android obliga a implementar en una clase dos métodos:

- **`onSensorChanged`** Para definir qué hacer cuando cambie algún valor en el sensor. En el caso del sensor de orientación, proporcionará dichos valores en un `array` con tres posiciones:

- **Posición 0** Para los grados `azimuth`.
- **Posición 1** Para los grados `pitch`.
- **Posición 2** Para los grados `roll`.

- **`onAccuracyChanged`** Para definir qué hacer cuando cambien la precisión del sensor.

Para implementar la rotación de mapa bastó con implementar el método `onSensorChanged` y pasar al mapa dicha rotación (ver listado 5.7).

```
public void onSensorChanged(SensorEvent event) {
    float azimuth = event.values[0];
    mMap.setMapOrientation(-azimuth);
}
```

Listado 5.7: Ejemplo de rotación de mapa en función del sensor de orientación

Pruebas

Al finalizar la implementación del tercer prototipo se realizaron las siguiente pruebas:

- Para comprobar la correcta orientación del mapa

Dado	Aplicación ejecutándose mientras se camina en dirección Norte Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se gira al Oeste o al Este
Entonces	Se muestra un giro a la izquierda y derecha respectivamente

Dado	Aplicación ejecutándose mientras se camina en dirección Sur Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se gira al Oeste o al Este
Entonces	Se muestra un giro a la derecha e izquierda respectivamente

5.3.4 Iteración 4: Crear y mostrar ruta

Una vez evaluado el prototipo de la iteración 3, se procedió a añadir una nueva funcionalidad: crear rutas para peatones, ciclistas o motoristas desde la posición actual hasta una determinada localización geográfica (latitud y longitud) y dibujarla en el mapa.

Diseño

Para poder calcular una ruta hay que hacer uso de alguno de los proveedores vistos en la sección 3.5. Puesto que se optó por usar los mapas de OSM y ya se tienen importadas las librerías *Osmdroid* y *Osmbonuspack* que permiten calcular rutas para peatones, ciclistas y motorias; la opción más lógica es utilizar *MapQuest*.

Implementación

Para la creación y superposición en el mapa de una ruta utilizando *Osmdroid* y *Osmbonuspack* basta con indicar el tipo de usuario, el punto de inicio de la ruta y de finalización. Con esta información, realiza una consulta en Internet a la API de *MapQuest* [2] y devuelve una ruta específica que solo resta dibujar en el mapa (ver listado 5.8).

```
private Boolean gotoGeoPoint(GeoPoint endPoint) {
    RoadManager roadManager = new MapQuestRoadManager(MAPQUESTAPIKEY);
    roadManager.addRequestOption("units=k");
    roadManager.addRequestOption("routeType=bitycle");

    ArrayList<GeoPoint> waypoints = new ArrayList<GeoPoint>();
    waypoints.add(getLastLocation());
    waypoints.add(endPoint);

    mRoad = roadManager.getRoad(waypoints);

    mRoadOverlay=RoadManager.buildRoadOverlay(road, getBaseContext());
    mRoadOverlay.setWidth(10);

    mMap.getOverlays().add(mRoadOverlay);
    mMap.invalidate();
}
```

Listado 5.8: Ejemplo de creación y superposición de una ruta en el mapa

Pero existe un problema con el listado 5.8: Cuando se utiliza como objetivo de desarrollo una API de Android superior a 9, no realiza la petición a Internet y muestra una línea recta entre el origen y el destino. Este *bug* esta documentado en *Osmbonuspack* [3] aunque no sea culpa de la librería.

El problema radica en la forma que se realiza la petición a Internet. Desde la API de Android 10 y posteriores, Google decidió que realizar peticiones a Internet en el mismo *thread* en el que se ejecuta la Graphical User Interface (GUI) era un error. Lo estimó así porque podía provocar la sensación de que la aplicación no funciona cuando lo que verdaderamente ocurre es que se encuentra esperando una respuesta de Internet.

Por tanto, para que las rutas funcionen sólo se pueden realizar dos acciones:

- Eliminar las políticas de *threads* (ver listado 5.9) y seguir usando el mismo código.
- Utilizar un *thread* diferente al de la GUI para realizar la petición a Internet. (ver listado 5.10.

[2] http://wiki.openstreetmap.org/wiki/MapQuest
[3] https://code.google.com/p/osmbonuspack/issues/detail?id=9

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
    .permitAll().build();
StrictMode.setThreadPolicy(policy);
```

Listado 5.9: Eliminación de políticas de threads

```
private void gotoGeoPoint(GeoPoint endPoint) {
     ArrayList<GeoPoint> waypoints = new ArrayList<GeoPoint>();
     waypoints.add(getLastLocation());
     waypoints.add(endPoint);

     new GetRoad().execute(waypoints);
}

private class GetRoad extends AsyncTask<ArrayList<GeoPoint>, Float,
    Boolean>{
     protected Boolean doInBackground(ArrayList<GeoPoint>... params) {
          RoadManager roadManager = new MapQuestRoadManager(
              MAPQUESTAPIKEY);
          roadManager.addRequestOption("units=k");
          roadManager.addRequestOption("routeType=" + mTransport);

          mRoad = roadManager.getRoad(params[0]);

          return true;
     }
     protected void onPostExecute(Boolean result) {
          if (result) gotoGeoPointPosThread();
  }

private void gotoGeoPointPosThread() {
     mRoadOverlay = RoadManager.buildRoadOverlay( mRoad,
       getBaseContext() );
     mRoadOverlay.setWidth(10);

     mMap.getOverlays().add(mRoadOverlay);
     mMap.invalidate();
}
```

Listado 5.10: Ejemplo de creación y superposición de una ruta en el mapa utilizando una `AsyncTask`

Para implementarlo en el sistema, se seleccionó la segunda opción porque se consideró que era la forma correcta de hacerlo y se añadieron algunas animaciones (no incluidas en el listado 5.10) para la espera en la recepción de la ruta.

Pruebas

Al finalizar la implementación del cuarto prototipo se realizó la siguiente prueba:

- Para comprobar el correcto cálculo de rutas para diferentes usuarios y destinos

Dado	Diferentes perfiles de usuarios: peatones, ciclistas y motoristas Diferentes coordenadas de destinos Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se llama al método `gotoGeoPoint`
Entonces	Se observa en el mapa la ruta desde nuestra localización hasta las coordenadas elegidas teniendo en cuenta el método de desplazamiento

5.3.5 Iteración 5: Navegar por ruta

Una vez evaluado el prototipo de la iteración 4, se procedió a añadir una nueva funcionalidad: navegar por la ruta creada en la iteración anterior observando los mensajes suministrados por el *smartphone*.

Diseño

Para diseñar esta nueva funcionalidad se tuvieron en cuenta tres factores:

- La **velocidad** a la que se desplaza el usuario. Hay que tener en cuenta que, cuando se hace uso de la aplicación, el tiempo con el que hay que avisar para realizar una acción variará en función de la velocidad que se lleve. Aprovechando este punto, también se incorpora una mejora visual que tienen la mayoría de aplicaciones de navegación: cambiar el *zoom* del mapa en función de la velocidad que se lleve para que no sufra grandes cambios a la hora de circular a una velocidad elevada.

- Cómo **detectar si se sigue la ruta marcada**. Hay que tener en cuenta que sólo se dispone de las coordenadas geográficas del dispositivo y de las coordenadas del siguiente punto a alcanzar. Si se implementa un algoritmo que permita pasar a la siguiente etapa cuando se llegue a la que se encuentra realizando, es posible que no funcione. Por ejemplo, si un usuario se acerca a una etapa de *girar a la derecha* y gira en el sitio correcto antes de que la aplicación detecte que ha llegado (probablemente por la precisión de los satélites GPS), no cargaría la siguiente etapa y no sabría qué está sucediendo.

- El posible **desvío de la ruta establecida**. Hay que plantearse cómo detectar las salidas de la ruta marcada y qué hacer en esos casos. Al tratarse de una aplicación de navegación diseñada para que ella interactúe con el usuario y no al contrario, la opción más interesante es que, cuando detecte el desvío, avise al usuario y calcule una nueva ruta desde la posición en la que se encuentre.

Implementación

Las tres implementaciones antes descritas, se llevaron a cabo dentro del método onLocationChanged que ya se comenzó a escribir en la iteración 2. Se debe realizar en éste método porque, tal y como está configurado en el LocationManager (ver listado 5.4), se ejecutará cada segundo con los nuevos datos suministrados por los satélites GPS:

- En función de la **velocidad**, se consideraron validos los valores de cercanía y *zoom* reflejados en el listado 5.11.

```
public void onLocationChanged(Location loc) {
  [...]

  float speed = loc.getSpeed();
  int metersToFirstWarning;

  if (speed > kmhToMs(100)) {
      mMap.getController().setZoom(15);
      metersToFirstWarning = 1000;
  } else if (speed > kmhToMs(80)) {
      mMap.getController().setZoom(16);
      metersToFirstWarning = 500;
  } else if (speed > kmhToMs(50)) {
      mMap.getController().setZoom(17);
      metersToFirstWarning = 100;
  } else if (speed > kmhToMs(20)) {
      mMap.getController().setZoom(18);
      metersToFirstWarning = 60;
  } else {
      mMap.getController().setZoom(18);
      metersToFirstWarning = 30;
  }
```

Listado 5.11: Ejemplo de implementación de LocationListener utilizado para variar el zoom del mapa y determinar la distancia necesaria para avisar de las acciones

- Para **detectar si se sigue la ruta marcada** se ha implementado el algoritmo del listado 5.12. El algoritmo indica la acción de *seguir recto* hasta que la distancia hasta la próxima etapa es menor que la distancia del comienzo de los avisos del punto anterior. Es entonces cuando:
  - Si se acerca, se muestra la etapa a realizar.
  - Si se aleja, se considera superada la etapa a realizar. De este modo, se pasa a la siguiente etapa, a menos que sea la última y finalice la navegación.

```
private Boolean mNewAction = true;
private int mStep = 0;
private float mMetersToGoal = Float.MAX_VALUE;

public void onLocationChanged(Location loc) {
  GeoPoint myLocation = new GeoPoint(loc);

  [...]

  if (mRoad != null) {
    float distance = getDistance(myLocation, mRoad.mNodes.get(
        mStep).mLocation);

    if (distance < metersToFirstWarning) {
      if (distance < mMetersToGoal) {
        if (mNewAction || mMetersToGoal == Float.MAX_VALUE) {
          mNewAction = false;
          startAction(getAction(mRoad.mNodes.get(mStep).
              mManeuverType), distance);
        }
        mMetersToGoal = distance;
      } else {
        mNewAction = true;
        mMetersToGoal = Float.MAX_VALUE;

        if (mStep == mRoad.mNodes.size()-1) {
          cleanMap();  //Finished
        } else {
          mStep++;
        }
      }
    } else {
      if (mNewAction) {
        mNewAction = false;
        startAction(STRAIGHT, distance);
      }
    }
  }
```

Listado 5.12: Ejemplo de implementación de `LocationListener` utilizado para seguir una ruta almacenada en `mRoad`

- Para detectar cuándo se **desvía de la ruta establecida** se ha utilizado una función de la clase `RoadOverlay` que indica en píxeles a cuanta distancia se encuentra un punto de una ruta (ver listado 5.13). Puesto que la ruta va a tener siempre la misma anchura en píxeles tanto acercándose como alejándose de ella con el *zoom* y el `LocationOverlay` también (24 píxeles), con éste método se puede determinar cuándo se abandona la ruta establecida y coincidirá con lo expresado en la pantalla.

  Además para prevenir posibles errores puntuales en la localización, para considerar un desvío de ruta, se ha implementado un contador de errores y sólo se vuelve a calcular el recorrido una vez que se ha detectado 3 veces el desvío. Como las medidas tardan un segundo en realizarse (ver listado 5.4), se tardará 3 segundos en detectar el error.

```
private int mRoadLost = 0;

public void onLocationChanged(Location loc) {
  GeoPoint myLocation = new GeoPoint(loc);

  [...]

  if (mRoadOverlay != null &&
      !mRoadOverlay.isCloseTo(myLocation, 24, mMap)) {
    if (mRoadLost < 3) {
      mRoadLost++;
    } else {
      startAction(WRONG, 0);
      GeoPoint endPoint =
        mRoad.mNodes.get(mRoad.mNodes.size()-1).mLocation;
      cleanMap();
      gotoGeoPoint(endPoint);
    }
  }
```

Listado 5.13: Ejemplo de implementación de `LocationListener` utilizado para detectar cuándo se desvía de la ruta establecida

Pruebas

Al finalizar la implementación del quinto prototipo se realizaron las siguientes pruebas:

- Para comprobar las diferentes configuraciones de *zoom* y distancia para avisar en función de la velocidad.

Dado	Diferentes velocidades
	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se pasa entre los diferentes rangos del listado 5.11
Entonces	Cambia el zoom del mapa

Dado	Diferentes velocidades
	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se acerca una acción
Entonces	La distancia con la que avisa varia en función de la velocidad

- Para comprobar el correcto funcionamiento del guiado entre etapas

Dado	Diferentes rutas Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se navega por una ruta
Entonces	Muestra cada una de las etapas hasta el final

- Para comprobar el correcto funcionamiento del sistema cuando el usuario se pierde

Dado	Diferentes rutas Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se desvía de una ruta y pasan 3 segundos
Entonces	Avisa del desvío y recalcula la ruta a seguir desde la posición actual hasta la meta fijada en un inicio

5.3.6 Iteración 6: Selector de destino

Una vez evaluado el prototipo del a iteración 5, se procedió a añadir una nueva funcionalidad: seleccionar el destino de nuestra ruta especificando la dirección deseada y no sus coordenadas geográficas.

Diseño

Para determinar las coordenadas geográficas de un sitio introduciendo su nombre hay que hacer uso de alguno de los proveedores de *geocoding* estudiados en la sección 3.5. Como se está desarrollando todo el proyecto con OSM lo más normal sería hacer uso de *Nominatim* por medio de la librería *Osmbonuspack* pero existe una opción mejor: usar la API de Google que ofrece gratuitamente 2.500 peticiones. Y además, permite realizar búsquedas no sólo por ciudades y calles como *Nominatim*, sino que también es posible buscar a nivel de números de casa.

Implementación

Para implementar esta nueva funcionalidad hay que desarrollar una nueva `Activity`. Ésta será la encargada de preguntar al servicio de geocoding por la dirección introducida, permitir seleccionar al usuario entre las diferentes respuestas del servicio y devolver al `Activity` principal las coordenadas geográficas y el tipo de transporte seleccionado.

Para hacer uso en Android del geocoding de Google, sólo es necesario declarar un objeto del tipo `Geocoder` pero, al igual que en la cuarta iteración, hay que declararlo dentro de una `AsyncTask` porque hace uso de Internet (Ver listado 5.14).

```java
private List<Address> foundAdresses;

private class GetAddresses extends AsyncTask<Void, Float, Boolean> {
  protected Boolean doInBackground(Void... params) {
    Geocoder geocoder = new Geocoder(getApplicationContext(),
        new Locale(getString(R.string.locale)));
    String text = search_text.getText().toString();

    try {
      foundAdresses = geocoder.getFromLocationName(text, 20);
      return true;
    } catch (IOException e) {
      return false;
    }
  }

  protected void onPostExecute(Boolean result) {
    if (result) getAddressesPost();
  }
}
```

Listado 5.14: Ejemplo de implementación de `Geocoder` dentro de una `AsyncTask`

Pruebas

Al finalizar la implementación del sexto prototipo se realizaron las siguientes pruebas:

- Para comprobar el correcto funcionamiento del servicio de *geocoding*

Dado	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se introducen diferentes nombres de lugares de diferentes ciudades
Entonces	La nueva actividad muestra las direcciones de los lugares buscados

- Para comprobar el correcto funcionamiento del paso de información entre actividades

Dado	Diferentes nombres introducidos en la nueva actividad Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Se selecciona algún resultado de la búsqueda
Entonces	La aplicación guía desde la ubicación actual hasta el lugar seleccionado

5.3.7 Iteración 7: Avisos por pantalla

Una vez evaluado el prototipo de la iteración 6 se introdujo una nueva funcionalidad: mostrar los avisos de la pantalla de forma que estuvieran integrados dentro de la aplicación y no como hasta ahora que se muestran por medio de mensajes `Toast`.

Diseño

Puesto que se conocen las acciones que el usuario debe realizar y cuándo (ver listado 5.12), sólo hay que modificar la apariencia de la aplicación y añadir el código necesario para realizar los cambios visuales en el método `startAction`.

Para modificar la apariencia de la aplicación se añadió un panel en la parte inferior de la pantalla. La parte izquierda del panel muestra una imagen de la próxima acción a realizar y la distancia que falta para realizarla. La parte derecha del panel muestra textualmente la acción que se debe realizar.

Implementación

Para modificar la apariencia de la aplicación basta con añadir al `layout` de la actividad principal los paneles del listado 5.15. Las modificaciones en el método `startAction` son poco significativas.

Pruebas

Al finalizar la implementación del séptimo prototipo se realizó la siguiente prueba:

- Para comprobar que se muestran las acciones por pantalla

Dado	Navegación en curso hacia algún lugar
	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Es necesario realizar alguna acción
Entonces	Se muestra por pantalla la acción a realizar

5.3.8 Iteración 8: Avisos sonoros

Una vez evaluado el prototipo de la iteración 7 se introdujo una nueva funcionalidad: escuchar los avisos que se muestran por pantalla.

Diseño

Al igual que en la iteración anterior, como ya se conocen las acciones a realizar y cuándo (ver listado 5.12) sólo hay que añadir el código necesario para escuchar las instrucciones en el método `startAction`.

Hay dos posibles formas de escuchar las instrucciones a realizar:

- Tener todas las instrucciones grabadas y reproducirlas en los momentos adecuados.
- Pedir a Internet que envíe el sonido de un texto, en este caso la acción a realizar, y reproducirla. Este servicio se llama *text to speech*.

Puesto que existen muchos mensajes diferentes, ya es necesaria la conexión a Internet para recibir los mapas y los mensajes se pueden personalizar mucho más con la segunda opción; la elección que se tomó para reproducir sonoramente las instrucciones fue *text to speech*. Y se eligió el servicio de Google como proveedor por su gran integración con Android.

Implementación

Para convertir texto a sonido con el servicio de Google solamente hay que instanciar una clase del tipo `TextToSpeech` especificando el contexto y qué clase implementa la función `onInit`. Después implementar dicha función indicando el lenguaje deseado y llamar al método `speak`.

En el ejemplo del listado 5.16 se ha simplificado el uso y se han prevenido errores creando el método `convertTextToSpeech`. De esta forma, cuando se desee reproducir sonoramente un texto sólo hay que llamar a dicho método.

```
private TextToSpeech mTextToSpeech = new TextToSpeech(this, this);

[...]

public void onInit(int status) {
  if (status == TextToSpeech.SUCCESS) {
    int result = mTextToSpeech.setLanguage(new Locale("es"));
    if (result == TextToSpeech.LANG_MISSING_DATA ||
        result == TextToSpeech.LANG_NOT_SUPPORTED) {
      Toast.makeText(getBaseContext(),"Error al convertir el sonido",
          Toast.LENGTH_SHORT).show();
    } else {
      convertTextToSpeech("");
    }
  }
}

private void convertTextToSpeech(String text) {
  if (!text.equals("") && mTextToSpeech != null) {
    mTextToSpeech.speak(text, TextToSpeech.QUEUE_FLUSH, null);
  }
}
```

Listado 5.16: Ejemplo del uso de `TextToSpeech`

Pruebas

Al finalizar la implementación del octavo prototipo se realizó una prueba similar a la de la iteración anterior:

- Para comprobar que se oyen las acciones a realizar

Dado	Navegación en curso hacia algún lugar
	Diferentes dispositivos con distintas versiones de Android (ver sección 4.2.1)
Cuando	Es necesario realizar alguna acción
Entonces	Se oye la acción a realizar

5.3.9 Iteración 9: Avisos vibratorios

Una vez evaluado el prototipo de la iteración 8 se introdujo una nueva funcionalidad: integrar los avisos por medio de vibraciones.

Diseño

Para guiar a un peatón, ciclista o motorista por una ruta específica existen varios tipos de acciones a realizar:

- Girar a la izquierda
- Girar a la derecha
- Continuar recto
- Dar media vuelta
- Camino equivocado
- Llegar al destino seleccionado
- En la rotonda, tomar una salida en específico. Por ejemplo la 3º, que no tiene que coincidir con girar a la izquierda.

Puesto que se trata de un número elevado de acciones para interpretar con un único vibrador se pensó en utilizar un complemento vibratorio y codificar todas las posibles acciones en intuitivas instrucciones descritas en el cuadro 5.1.

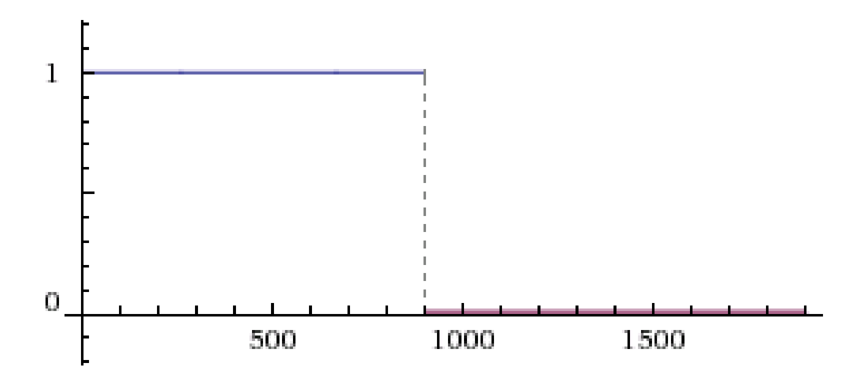


Figura 5.1: Frecuencia de vibración para un giro. El eje vertical indica si está (1) o no (0) activo el vibrador, y el eje horizontal indica el tiempo en milisegundos

Acción	**Codificación**
Girar a la izquierda	Vibración continua del dispositivo situado a la izquierda utilizando la frecuencia representada en la figura 5.1
Girar a la derecha	Vibración continua del dispositivo situado a la derecha utilizando la frecuencia representada en la figura 5.1
Continuar recto	Sin vibración
Dar media vuelta	Vibración continua hasta que se realice la acción
Camino equivocado	Vibración continua durante 5 segundos
Llegar al destino seleccionado	Repetición en tres ocasiones de la frecuencia representada en la figura 5.2
En la rotonda, 1º salida	Vibración continua de ambos dispositivos utilizando la frecuencia representada en la figura 5.3
En la rotonda, 2º salida	Vibración continua de ambos dispositivos utilizando la frecuencia representada en la figura 5.4
En la rotonda, 3º salida	Vibración continua de ambos dispositivos utilizando la frecuencia representada en la figura 5.5
En la rotonda, otra salida	Vibración continua de ambos dispositivos utilizando una frecuencia deducible con las figuras 5.3, 5.4 y 5.5.

Cuadro 5.1: Codificación de instrucciones

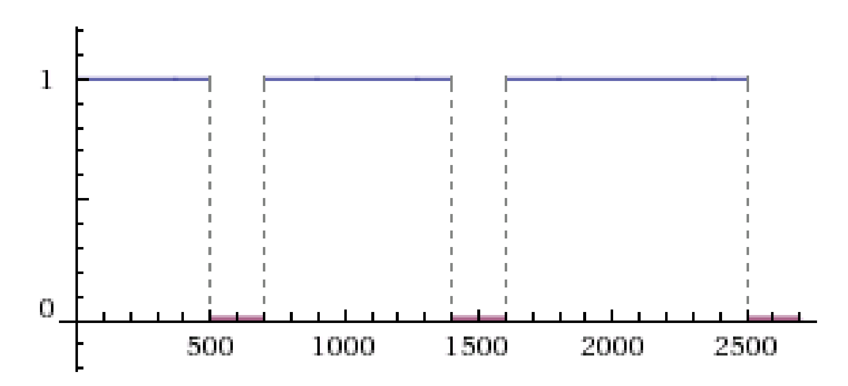


Figura 5.2: Frecuencia de vibración que determina si ha llegado al destino. El eje vertical indica si está (1) o no (0) activo el vibrador, y el eje horizontal indica el tiempo en milisegundos

Una vez determinada la forma de codificar las posibles acciones de los complementos vibratorios, hay que seleccionar qué complementos se utilizarán. En este punto surgió un conflicto de intereses. Tal y se vió en la sección 3.4.1 los únicos complementos que permiten controlar su vibrador a voluntad son los que llevan Android Wear como SO, pero estos dispositivos son muy caros y prácticamente nadie dispone de uno. Por ello, se propuso desarrollar el proyecto para poder utilizar tanto *wearables* con Android Wear como otros dispositivos con Android. De este modo podrán utilizar el sistema:

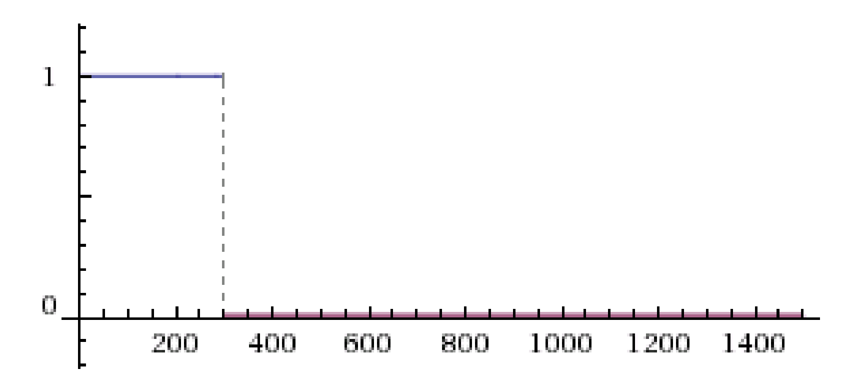


Figura 5.3: Frecuencia de vibración para tomar la primera salida de la rotonda. El eje vertical indica si está (1) o no (0) activo el vibrador, y el eje horizontal indica el tiempo en milisegundos

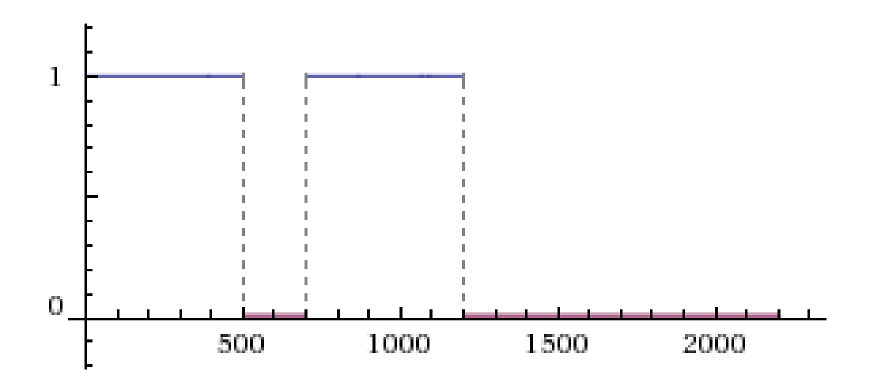


Figura 5.4: Frecuencia de vibración para tomar la segunda salida de la rotonda. El eje vertical indica si está (1) o no (0) activo el vibrador, y el eje horizontal indica el tiempo en milisegundos

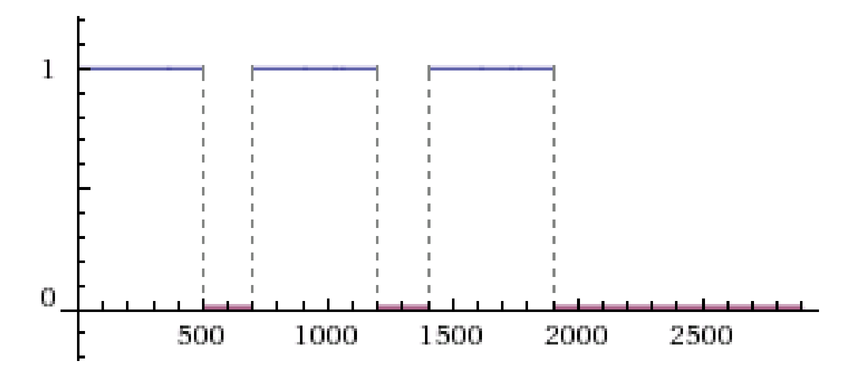


Figura 5.5: Frecuencia de vibración para tomar la tercera salida de la rotonda. El eje vertical indica si está (1) o no (0) activo el vibrador, y el eje horizontal indica el tiempo en milisegundos

- Los usuarios de **Android Wear** por medio de su *smartphone* y su wearable. En el teléfono funcionará la aplicación de navegación y se comunicará con el complemento para que vibre en los momentos apropiados.

- Los usuarios que dispongan de **dos dispositivos con Android**. En uno funcionará la aplicación de navegación y se comunicará con la aplicación del segundo para vibrar en los momentos oportunos.
- Los usuarios que dispongan de **un dispositivo Android** en el que se ejecute la aplicación principal y **dos complementos** que vibren en los momentos adecuados. Los complementos pueden ser:
  - Dos dispositivos con Android
  - Dos dispositivos con Android Wear
  - Un dispositivo con Android y otro con Android Wear

Para soportar este gran número de configuraciones antes expuesta se hace uso de la arquitectura cliente servidor por medio de la tecnología *bluetooth*:

- El **cliente** es la aplicación principal que implementa el sistema de navegación y se conecta a los demás dispositivos para hacerlos vibrar de la forma que desee (ver cuadro 5.1) y parar la vibración cuando estime oportuno.
- Los **servidores** serán los diferentes dispositivos con Android o Android Wear que quedarán a la espera de recibir instrucciones del cliente.

De este modo sólo hay que implementar en la aplicación de las anteriores iteraciones las vibraciones del cuadro 5.1, crear una nueva aplicación para Android, otra para Android Wear que también implemente dichas vibraciones y desarrollar un método de paso de mensajes por medio de *bluetooth* entre las dos aplicaciones.

Implementación

Gracias a que Android y Android Wear comparten gran parte de las API se puede utilizar el mismo código para la parte de la vibración en el cliente y en el servidor. Para codificar el cuadro 5.1 se hizo uso de la clase `Vibrator` que, tras inicializarla, permite manipular el vibrador a voluntad con el método `vibrate`. De este modo se desarrollaron los métodos `startLocalVibration` y `stopLocalVibration` (ver listado 5.17).

Por desgracia, para establecer la comunicación por *bluetooth* entre el cliente y los servidores no se puede utilizar el mismo método en Android y Android Wear. Android no se puede vincular a otro dispositivo como un *wearable* por medio de los *Play Services* de Android y Android Wear da muchos problemas para acceder a la *interfaz bluetooth* y comunicarse por otro método diferente a los *Play Services*. Por ello hay que distinguir los dos tipos de comunicación: Android-Android y Android-Android Wear.

Implementación de la comunicación con Android

Para establecer una comunicación por *bluetooth* entre el cliente y los servidores Android se utilizó la clase `BluetoothChatService` distribuida en los ejemplos de Android[4]. Esta clase permite abstraerse de la implementación realizada con `BluetoothSocket` y centrarse en qué hacer cuando cambia de estado (*escuchando*, *conectando*, etc) o recibe un mensaje. Además permite iniciar el «chat» y enviar mensajes de forma sencilla.

En la aplicación **cliente**, que es la que se desarrolló en las iteraciones anteriores, se añadió un nuevo menú en opciones. Este menú permite activar o desactivar la utilización de la vibración y, en caso de activarlo, permite seleccionar entre los diferentes dispositivos *bluetooth* emparejados y el dispositivo local cuál hará de vibrador derecho y cuál de vibrador izquierdo. Una vez seleccionado, el cliente intentará iniciar el «chat» con el/los servidor/es y se quedará a la espera. En caso de no conseguir establecer conexión simplemente avisará al usuario.

```
private BluetoothChatService mChatServiceLeft =
  new BluetoothChatService(this, mHandler);
private BluetoothChatService mChatServiceRight =
  new BluetoothChatService(this, mHandler);
private StringBuffer mOutStringBufferLeft = new StringBuffer("");
private StringBuffer mOutStringBufferRight =new StringBuffer("");

[...]

private void sendMessage(String message, int who) {
  if (who == LEFT) {
    sendMessage(message, mChatServiceLeft, mOutStringBufferLeft);
  } else if (who == RIGHT) {
    sendMessage(message,mChatServiceRight,mOutStringBufferRight);
  }
}
private void sendMessage(String message, BluetoothChatService d,
   StringBuffer b) {
  if (d == null ||
      d.getState() != BluetoothChatService.STATE_CONNECTED) {
    return;
  }

  message = message + SPLIT;
  if (message.length() > 0) {
      byte[] send = message.getBytes();
      d.write(send);
      b.setLength(0);
  }
}
```

Listado 5.18: Métodos utilizados para enviar mensajes por bluetooth haciendo uso de la clase `BluetoothChatServide`

[4] https://developer.android.com/resources/samples/BluetoothChat/index.html

Gracias a utilizar la clase `BluetoothChatServide`, para iniciar el intercambio de mensajes sólo es necesario iniciar el servicio con el método `start` y hacer uso del método `connect` proporcionándole el número Media Access Control (MAC) del dispositivo al que se quiere conectar. Para definir cómo se comporta la *interfaz gráfica* de la aplicación durante el proceso de conexión hubo que definir un `Handler` y sobrescribir el método `handleMesage` indicando qué hacer mientras se conecta, cuando se desconecte o cuando falle el intento de conexión. Una vez existe conexión se pueden enviar mensajes a los dispositivos conectados por *bluetooth* utilizando los métodos del listado 5.18.

Para el caso del **servidor** Android se desarrolló una aplicación que inicia el servicio `BluetoothChatService` al ejecutarse quedando a la escucha de conexiones *bluetooth*. Además se definió un `Handler` para especificar qué hacer en los diferentes estados de la conexión con la *interfaz gráfica* y determinar qué hacer en caso de recibir un mensaje (ver listado 5.19).

Como se puede apreciar en los listados 5.18 y 5.19 a la hora de intercambiar mensajes se incluyó un separador: `SPLIT`. Esto se debe a que durante el desarrollo del intercambio de mensajes se detectaron errores se sincronización entre los `buffers` del cliente y de los servidores. Este es el modo de asegurarse de no malinterpretar los mensajes.

Implementación de la comunicación con Android Wear

Para establecer la comunicación *bluetooth* entre el cliente y los servidores Android Wear se utilizó el servicio de paso de mensajes ofrecido por los *Play Services* en la clase *MessageApi*. Esta clase permite centrarse en el envío y recepción de mensajes sin preocuparse de la conexión.

En la aplicación **cliente** desarrollada en iteraciones anteriores y en el apartado anterior, se añadió a la lista de dispositivos de las opciones todos los *wearables* sincronizados por los *Play Services*. Para hacer más intuitiva la selección se añadió un subtítulo debajo de los nombres de los dispositivos indicando si se trataba de un elemento *local*, *bluetooth* o *wear*.

Gracias a utilizar los *Play Services* para comunicarnos con un dispositivo vinculado sólo hay que encargarse de sobrescribir los métodos `onConnected` y `onConnectionSuspended`, inicializar la clase `GoogleApiClient` y conectarse a los servicios con el método `connect` de dicha clase. A partir de entonces se pueden enviar mensajes a un nodo conectado indicando su identificador y el tipo de mensaje. En el listado 5.20 se muestran los métodos utilizados para el envío de mensajes a un *wearable* aunque no se incluye la inicialización de las variables `mWearLeft` y `mWearRight` con los identificadores de los nodos correspondientes. Esta parte se realiza tras la selección de los dispositivos en las opciones.

```
private String mWearLeft = "";
private String mWearRight = "";

[...]

private void sendMessageWear(String text, int who) {
    if (who == LEFT) {
        sendMessageWear(WEAR_MESSAGE_PATH, text, mWearLeft);
    }else if (who == RIGHT) {
        sendMessageWear(WEAR_MESSAGE_PATH, text, mWearRight);
    }
}
private void sendMessageWear( final String path, final String
    text, final String nodeId) {
    new Thread( new Runnable() {
        @Override
        public void run() {
            Wearable.MessageApi.sendMessage(
              mApiClient, nodeId, path, text.getBytes()).await();
        }
    }).start();
}
```

Listado 5.20: Métodos utilizados para enviar mensajes por bluetooth haciendo uso de los Play Services

Para el caso del **servidor** en Android Wear se desarrolló un servicio y una aplicación. El servicio se encarga de iniciar la aplicación cuando recibe el tipo de mensaje correspondiente (ver listado 5.21) y la aplicación se encarga de interpretar el resto de mensajes recibidos para hacer vibrar el dispositivo o finalizar la aplicación. Para ello, en la aplicación se ha sobrescrito los métodos `onMessageReceived`, `onConnected` y `onConnectionSuspended` (ver listado 5.22); se inicializa la clase `GoogleApiClient` y se conecta a los servicios por medio del método `connect` de dicha clase.

```
public class WearMessageListenerService extends
    WearableListenerService implements Const{
  public void onMessageReceived(MessageEvent messageEvent) {
    if(messageEvent.getPath().equalsIgnoreCase(START_ACTIVITY)) {
      Intent intent = new Intent( this, MainActivity.class );
      intent.addFlags( Intent.FLAG_ACTIVITY_NEW_TASK );
      startActivity( intent );
    } else {
      super.onMessageReceived(messageEvent);
    }
  }
}
```

Listado 5.21: Servicio que inicia la aplicación en el wearable

```java
private GoogleApiClient mApiClient;
private Vibrator mVibrator;

[...]

@Override
public void onMessageReceived( final MessageEvent messageEvent )
    {
  runOnUiThread( new Runnable() {
    @Override
    public void run() {
      if(messageEvent.getPath().equalsIgnoreCase(
          WEAR_MESSAGE_PATH)) {
        startAction(new String(messageEvent.getData()));
      }
      if (messageEvent.getPath().equalsIgnoreCase( END_ACTIVITY )
          ) {
        if (mVibrator != null) {
          mVibrator.cancel();
        }
        finish();
      }
    }
  });
}

[...]

@Override
public void onConnected(Bundle bundle) {
    Wearable.MessageApi.addListener( mApiClient, this );
}

@Override
public void onConnectionSuspended(int i) {
    if (mVibrator != null) {
        mVibrator.cancel();
    }
}
```

Listado 5.22: Métodos utilizados en los servidores para leer los mensajes recibidos por medio de los Play Services

Pruebas

Al finalizar la implementación del noveno prototipo se realizó la siguiente prueba:

- Para determinar el correcto funcionamiento de la vibración y del paso de mensajes

Dado	Diferentes dispositivos con distintas versiones de Android
	Todas las formas descritas de conectar los dispositivos
	Todas las posibles acciones de guiado
Cuando	Hay que realizar alguna acción
Entonces	Vibra el dispositivo adecuando con la frecuencia correspondiente

```
<LinearLayout
  android:id="@+id/navPanel"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_alignParentBottom="true"
  android:background="#222222"
  android:orientation="horizontal"
  android:padding="10dp" >

    <LinearLayout
      android:id="@+id/navPanelLeft"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:paddingLeft="10dp"
      tools:ignore="UseCompoundDrawables" >

        <ImageView
          android:id="@+id/navPanelImage"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:contentDescription="@string/empy" />

        <TextView
          android:id="@+id/navPanelKm"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:gravity="center"
          android:textColor="#e6e6e6"
          android:text="@string/empy"
          android:textAppearance="?android:attr/textAppearanceSmall" /
             >

    </LinearLayout>

    <TextView
      android:id="@+id/navPanelText"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center"
      android:paddingLeft="10dp"
      android:paddingRight="10dp"
      android:text="@string/alert_waiting_for_gps"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:textColor="#e6e6e6" />

</LinearLayout>
```

Listado 5.15: Panel inferior de la pantalla de navegación

```
private void startLocalVibration(int action) {
  if (!mIsInVibration) {
    mIsInVibration = true;

    switch (action) {
      case LEFT:
      case RIGHT:
        long[] turn = {0, 900, 1000};
        mVibrator.vibrate(turn, 0);
        break;

      case WRONG:
        mVibrator.vibrate(5000);
        break;

      case UTURN:
        long[] uturn = {0, 100};
        mVibrator.vibrate(uturn, 0);
        break;

      case DESTINATION:
        long[] destination = new long[19];
        for (int i=0; i<3; i++) {
          destination[i*6+1] = 500;
          destination[i*6+2] = 200;
          destination[i*6+3] = 700;
          destination[i*6+4] = 200;
          destination[i*6+5] = 900;
          destination[i*6+6] = 200;
        }
        mVibrator.vibrate(destination, -1);
        break;

      case ROUNDABOUT1:  case ROUNDABOUT2:  case ROUNDABOUT3:
      case ROUNDABOUT4:  case ROUNDABOUT5:  case ROUNDABOUT6:
      case ROUNDABOUT7:  case ROUNDABOUT8:
        int exit = action-20; //roundabout=2X --> x=exit number
        long[] roundabout = new long[exit*2+1];
        for (int i=0; i<exit; i++) {
          if (i != 0) roundabout[i*2] = 200;
            roundabout[i*2+1] = 500;
          }
          roundabout[exit*2] = 1000;
          mVibrator.vibrate(roundabout, 0);
          break;
    }
  } else {
    stopLocalVibration();
    startLocalVibration(action);
  }
}
private void stopLocalVibration() {
  if (mIsInVibration) {
    mVibrator.cancel();
    mIsInVibration = false;
  }
}
```

Listado 5.17: Métodos usados para hacer vibrar el dispositivo con la clase `Vibrator`

```java
private Handler mHandler = new Handler() {
  public void handleMessage(Message msg) {
    switch (msg.what) {
      case BluetoothChatService.MESSAGE_STATE_CHANGE:
        switch (msg.arg1) {
          case BluetoothChatService.STATE_CONNECTING:
          case BluetoothChatService.STATE_LISTEN:
          case BluetoothChatService.STATE_NONE:
            [...]
            break;
          case  BluetoothChatService.STATE_DISCONNECTING:
            [...]
            break;
        }
        break;

      case BluetoothChatService.MESSAGE_READ:
        byte[] readBuf = (byte[]) msg.obj;
        String readMessages = new String(readBuf, 0, msg.arg1);
        String readMessage = "";

        for (int i=0; i<readMessages.length(); i++) {
          if (readMessages.charAt(i) == SPLIT) {
            startAction(readMessage);
            readMessage = "";
          } else {
            readMessage = readMessage + readMessages.charAt(i);
          }
        }
        break;

      [...]
    }
  }
};

private void startAction (String readMessage) {
  if (readMessage.equals(STOP)) {
    stopLocalVibration();
  } else if (readMessage.length() > 1) {
    stopLocalVibration();
    int action = Integer.parseInt(readMessage.substring(1));
    startLocalVibration(action);
  }
}
```

Listado 5.19: Métodos utilizados en los servidores para leer los mensajes recibidos por medio de la clase `BluetoothChatServide`

Capítulo 6

Resultados

En el siguiente capítulo se presentan los resultados obtenidos tras desarrollar el sistema. Para ello se hace un análisis de los recursos que consume y el coste económico de desarrollarlo. Además se especifica paso a paso un caso de uso concreto. Finalmente se proporciona la dirección del repositorio donde están contenidos todos los archivos del proyecto.

6.1 Recursos y costes

En esta sección se desglosarán los recursos y los costes empleados en el sistema. Por un lado se presentan las estadísticas del proyecto, por otro una estimación del coste de acuerdo a las condiciones actuales del mercado y, finalmente, algunas medidas de rendimiento del sistema.

6.1.1 Estadísticas

En la tabla 6.1 se muestra el número de lineas de código del proyecto *Naviganto* completo. Para contabilizar dichas líneas se ha empleado el comando `cloc`.

Lenguaje	**Archivos**	**Espacios en blanco**	**Comentarios**	**Líneas de código**
XML	44	85	9	3213
Java	16	448	324	2594
Bourne Again Shell	1	20	21	123
DOS Batch	1	24	2	64
IDL	1	2	0	15
Total	**63**	**579**	**356**	**6009**

Cuadro 6.1: Número de líneas de código fuente de Naviganto por lenguajes

6.1.2 Coste económico

El desarrollo del proyecto *Naviganto* empezó aproximadamente el día 3 de Noviembre de 2014 y terminó alrededor del 9 de Enero de 2015, es decir, unas 10 semanas de desarrollo. Con una dedicación media de 6 horas diarias y 5 días a la semana, da como resultado un total de 300 horas de trabajo. Todo ello sin contabilizar el tiempo dedicado a la elaboración del presente documento.

Tomando como referencia la media en España de unos 16 €/hora para los analistas y desarrolladores [1] y los elementos *hardware* necesarios (ver sección 4.2.1) para el desarrollo, se ha elaborado el cuadro 6.2 con una estimación del coste.

Recurso	**Cantidad**	**Coste**
Sueldo del programador (300 horas)	1	4800 €
Computador de trabajo	1	499 €
Smartphone de la aplicación principal	1	399 €
Smartphone de la aplicación bluetooth	2	60 €
Smartwatch de la aplicación wear	2	239 €
Total		**6296 €**

Cuadro 6.2: Desglose de costes del desarrollo de Naviganto

El coste obtenido es puramente informativo ya que para calcular el presupuesto real se deben tener en cuenta otros muchos factores. De todos modos, es de suponer que el coste del desarrollo de *Naviganto* se ajustará más al cuadro 6.2 que a la estimación ofrecida del comando `sloccount` (ver cuadro 6.3) que calcula el tamaño del proyecto en líneas de código y emplea el modelo COnstructive COst MOdel (COCOMO) para obtener la estimación del tiempo y los costes necesarios para desarrollar el sistema.

Total Physical Source Lines of Code (SLOC)	3,206
Development Effort Estimate, Person-Years (Person-Months) (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))	0.68 (8.16)
Schedule Estimate, Years (Months) (Basic COCOMO model, Months = 2.5 * (person-months**0.38))	0.46 (5.55)
Estimated Average Number of Developers (Effort/Schedule)	1.47
Total Estimated Cost to Develop (average salary = $56,286/year, overhead = 2.40)	$ 91,813

Cuadro 6.3: Estimación de los costes del desarrollo de Naviganto según sloccount

6.1.3 Profiling

El *profiling* o medida de rendimiento consiste en medir los tiempos que el sistema tarda en realizar las operaciones más críticas. En el caso de *Navignato* se han seleccionado las siguientes operaciones:

- Mostrar posición actual en el mapa
- Buscar destino cercano (~10 km)
- Iniciar navegación hacia un destino cercano (~10 km)

[1] http://www.tusalario.es/main/salario/comparatusalario

Puesto que las tres operaciones dependen tanto de la aplicación como de la conexión a Internet, se ha desglosado en el cuadro 6.4 la media de tiempo trascurrido tras efectuar cada operación 10 veces sobre los diferentes tipos de conexión.

Acción	**Wifi**	**3G**	**2G**
Mostrar posición	4,98 s	6,315 s	5,98 s
Buscar destino	0,231 s	0,407 s	1,171 s
Iniciar navegación	1,086 s	2,329 s	25,177 s

Cuadro 6.4: Desglose de tiempos de ejecución de Naviganto

Como se puede observar, para el caso *mostrar posición* existe muy poca diferencia entre las conexiones a Internet porque el tiempo de espera viene determinado por la conexión con el satélite GPS. Esto es debido a que *Naviganto* almacena en caché las imágenes del mapa de la última posición en la que se encontró y sólo hay que esperar la posición actual.

En cambio, para *buscar destino* e *iniciar navegación* existe una notable diferencia de tiempos de ejecución en función de los datos requeridos de Internet.

6.2 Caso de uso concreto

En este apartado se introducirá un caso de uso concreto del sistema funcionando por medio de imágenes. Además, se detallará cada imagen para dar a conocer el estado en que se encuentra el sistema en ese momento.

En este caso de uso, el usuario se encuentra en Campo de Criptana en la calle Virgen de Criptana nº 58 y desea ir caminando hasta la Plaza Mayor. Para ello, utiliza el *smartphone* y dos complementos vibratorios: Un *smartwatch* conectado por *Play Services* y un *smartphone* conectado por *bluetooth*.

6.2.1 Prerequisitos

La figura 6.1 muestra los componentes hardware empleados por el sistema para este ejemplo de caso de uso. Previo inicio del sistema, las aplicaciones se encuentran instaladas en los dispositivos del siguiente modo:

- En el Nexus 5 se encuentra la aplicación principal llamada *Naviganto*.
- En el LG G Watch R se encuentra la aplicación para *wearables* llamada *Naviganto-Wear*.
- En el Nexus One se encuentra la aplicación para hacer vibrar vía *bluetooth* otros dispositivos Android. Se llama *NavigantoBluetooth*.

Además, el usuario decide colocarse en la mano izquierda el *LG G Watch R* y en el bolsillo derecho del pantalón el *Nexus One*.



Figura 6.1: Hardware utilizado para el caso de uso concreto

6.2.2 Inicialización

La figura 6.2 muestra el aspecto de *Naviganto* tras iniciarse y la figura 6.3 expone las diferentes opciones desarrolladas en el menú lateral.

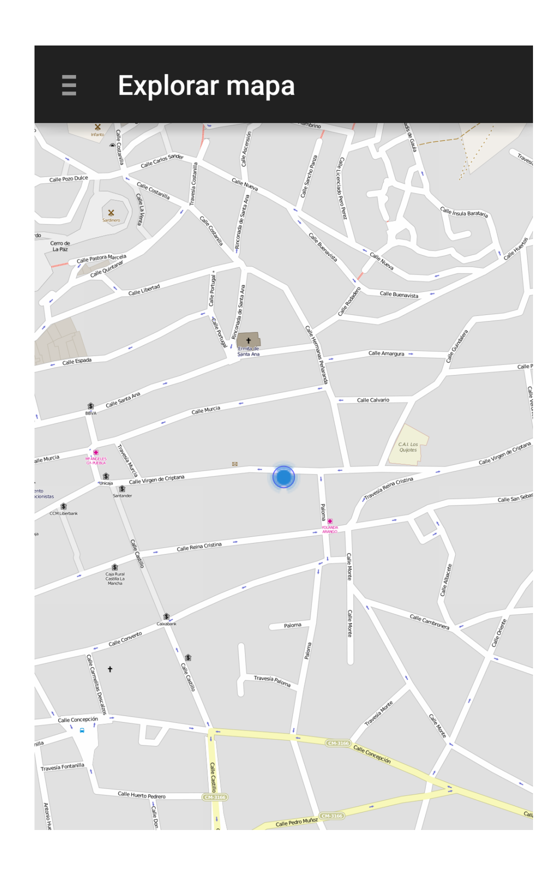


Figura 6.2: Naviganto tras iniciarse

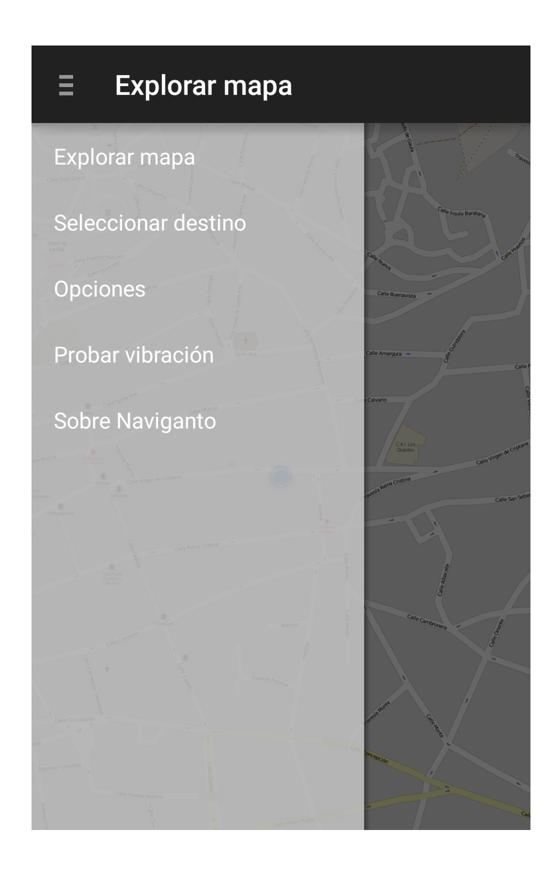


Figura 6.3: Barra lateral de Naviganto

6.2.3 Configuración de la vibración

Para configurar la vibración se selecciona en el menú lateral el apartado de *Opciones* y se muestra la figura 6.4 para configurar, entre otras alternativas, la vibración. Tras activar los *avisos vibratorios* (ver figura 6.5), se puede seleccionar el *Vibrador izquierdo* y el *Vibrador derecho*. Al tocar sobre cualquiera de ellos se muestra un listado de los posibles dispositivos (ver figura 6.6) y se puede proceder a dejar la configuración deseada (ver figura 6.7).

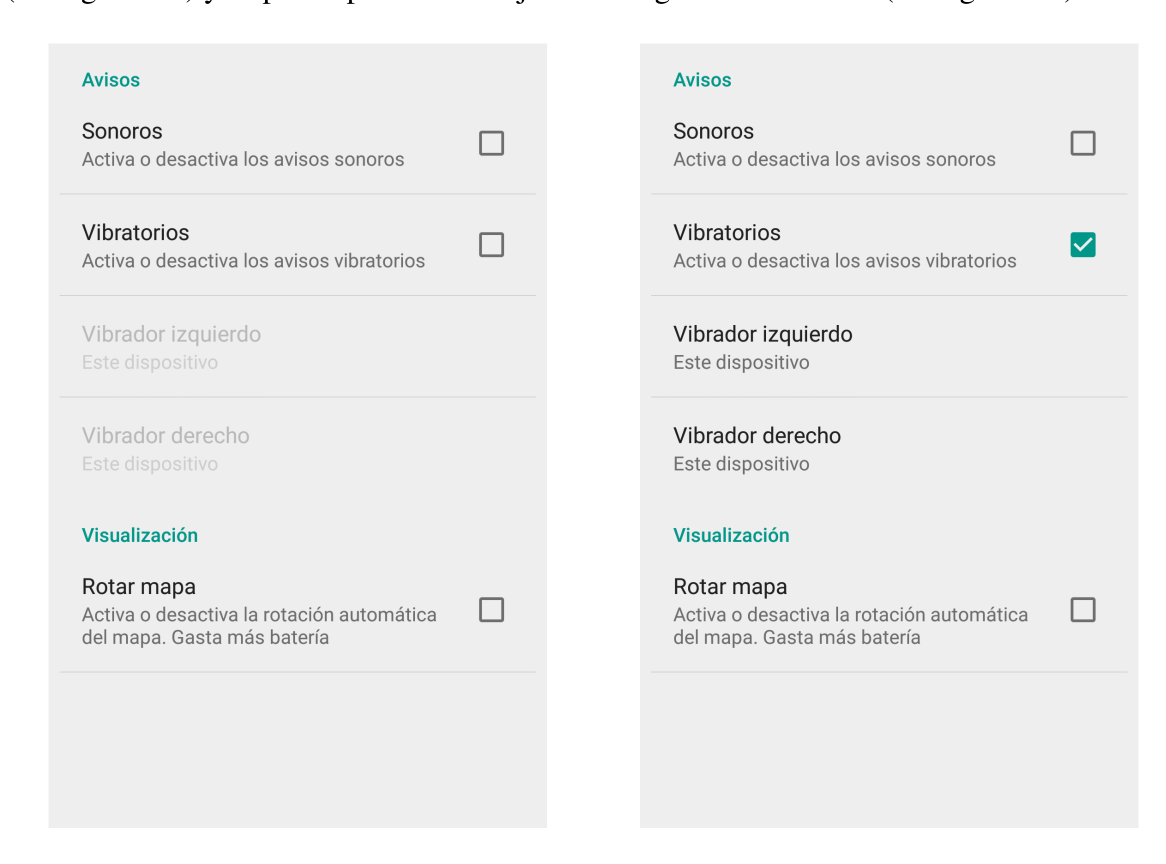


Figura 6.4: Opciones de Naviganto

Figura 6.5: Vibración activada

Este es el momento oportuno para ejecutar la aplicación *NavigantoBluetooth* del otro *smartphone* y que se quede a la espera (ver figura 6.8). También se puede ejecutar la aplicación *NavigantoWear* del *smartwatch* aunque no es necesario, esta aplicación puede iniciarse sola.

Ahora es el momento de pulsar la tecla de retroceso dentro de la opciones de *Naviganto* para que instantes después figuren los dos dispositivos como conectados (ver figuras 6.2.3 y 6.2.3).

Para comprobar que la configuración ha tenido éxito se selecciona la opción de la barra lateral de *Naviganto* llamada *Probar vibración* (ver Figura 6.3). De esta forma, se mandará a *NavigantoWear*, que está configurado en la posición izquierda, la orden de vibrar durante 0.9 segundos. Acto seguido se le mandará a *NavigantoBluetooth*, que está configurado en la posición derecha, la orden de vibrar durante otros 0.9 segundos. Gracias a esta comprobación se puede estar seguro de que la configuración de la vibración ha sido un éxito.



Figura 6.6: Selección de dispositivos

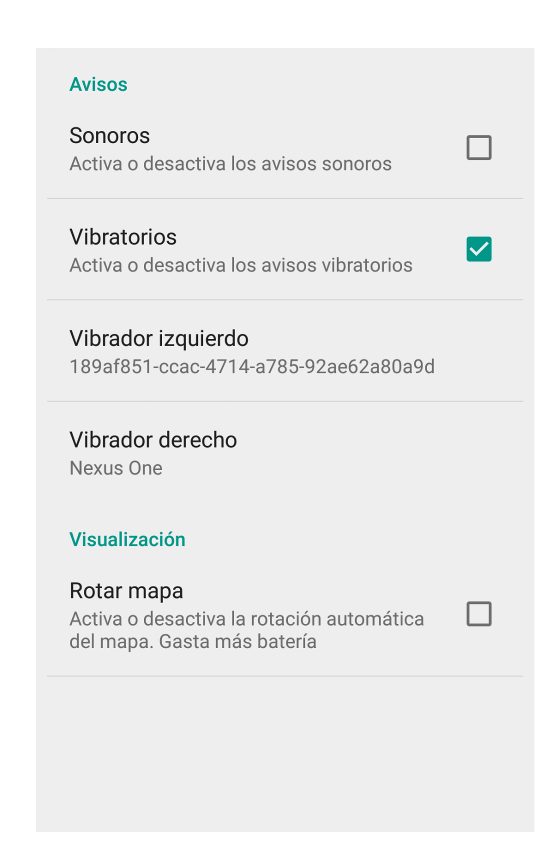


Figura 6.7: Opciones del caso de uso

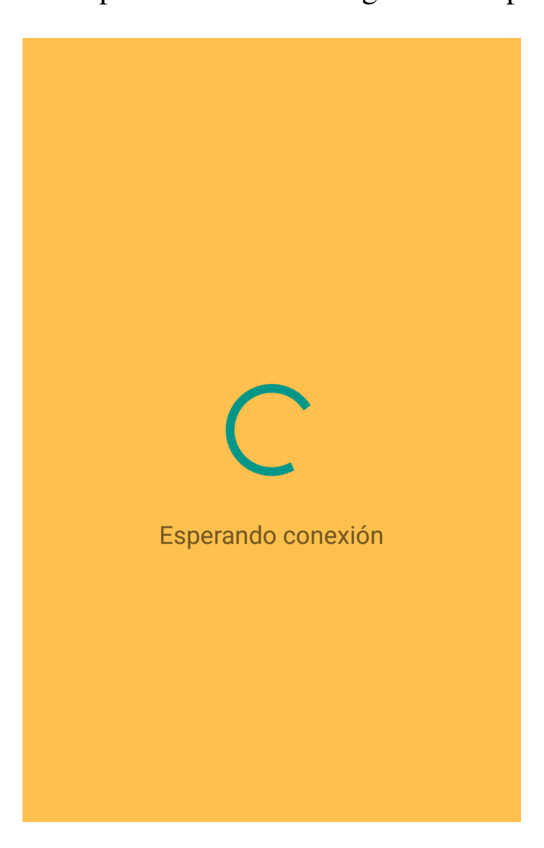


Figura 6.8: NavigantoBluetooth tras iniciarse, es decir, esperando conexión

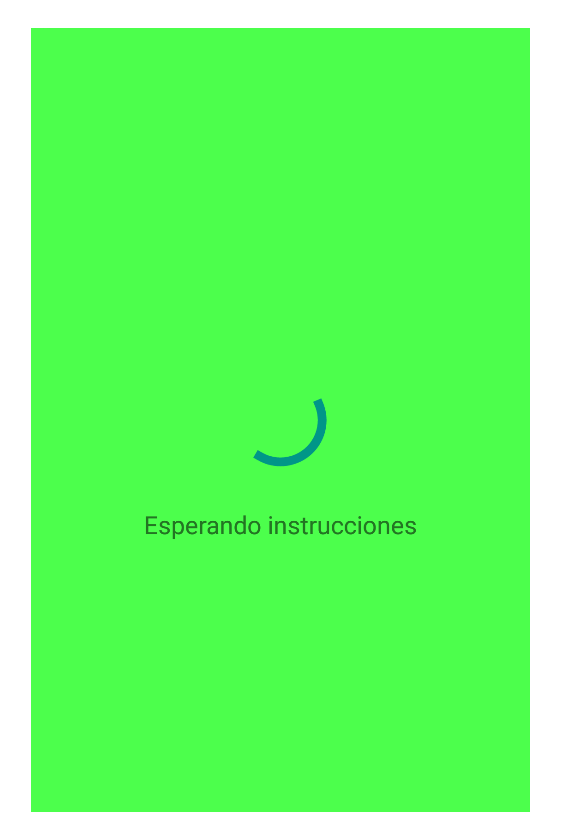



Figura 6.9: NavigantoBluetooth esperando instrucciones



Figura 6.10: NavigantoWear esperando instrucciones

6.2.4 La ruta

Para introducir el destino al que se desea ir, se elige la opción de la barra lateral *Seleccionar destino* tras lo cual aparecerá la figura 6.11. Después hay que seleccionar en el menú superior la opción *Peatón*, introducir la dirección y pulsar en el botón *Buscar* para que se muestre la figura 6.12. Finalmente, tras pinchar sobre el único resultado se iniciará la navegación (ver figura 6.13).

Cuando se inicie la marcha y se llegue al primer giro (ver figura 6.14) vibrará el *smartphone* del bolsillo derecho del pantalón con la frecuencia determinada para los giros (ver figura 5.1). Más tarde, cuando se llegue al segundo giro (ver figura 6.15) vibrará el *smartwatch* con la misma frecuencia. Finalmente, cuando se llegue al destino (ver figura 6.16) vibrarán ambos dispositivos con la frecuencia de *destino alcanzado* (ver figura 5.2).

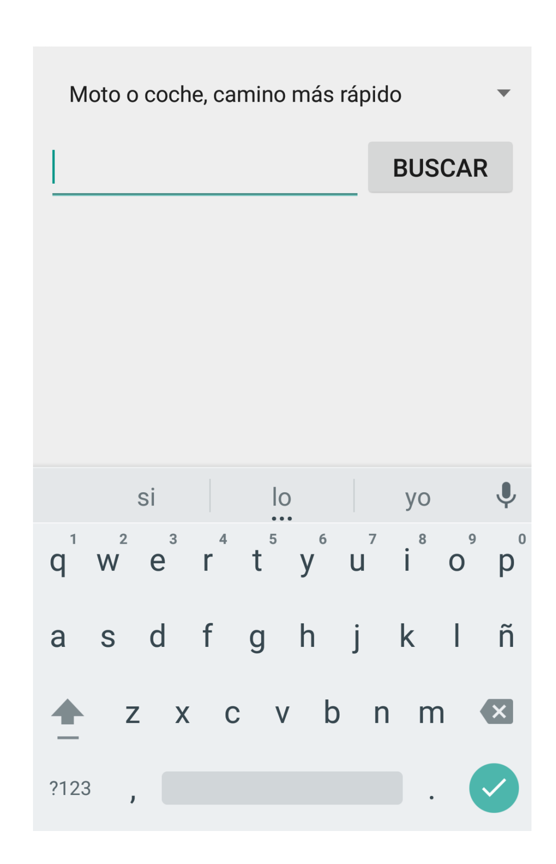


Figura 6.11: Buscador de destinos

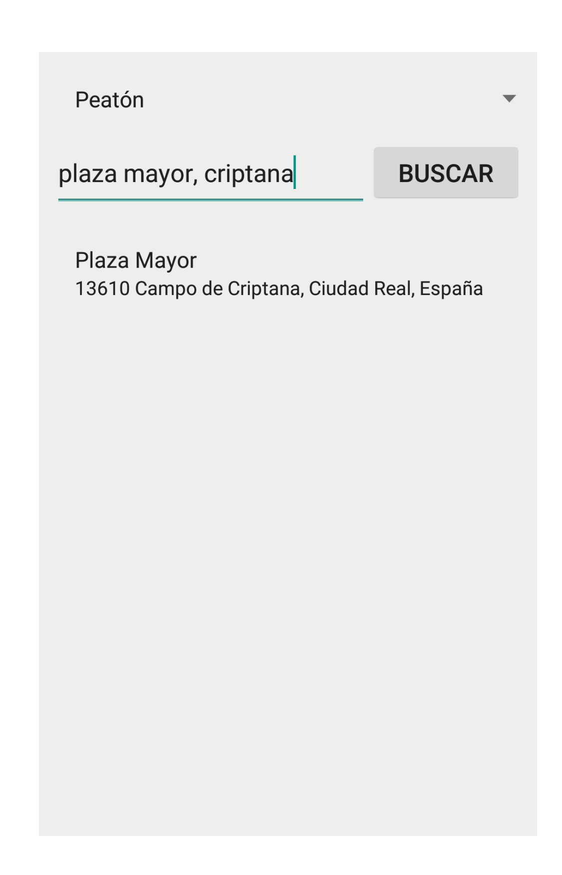


Figura 6.12: Búsqueda de destino

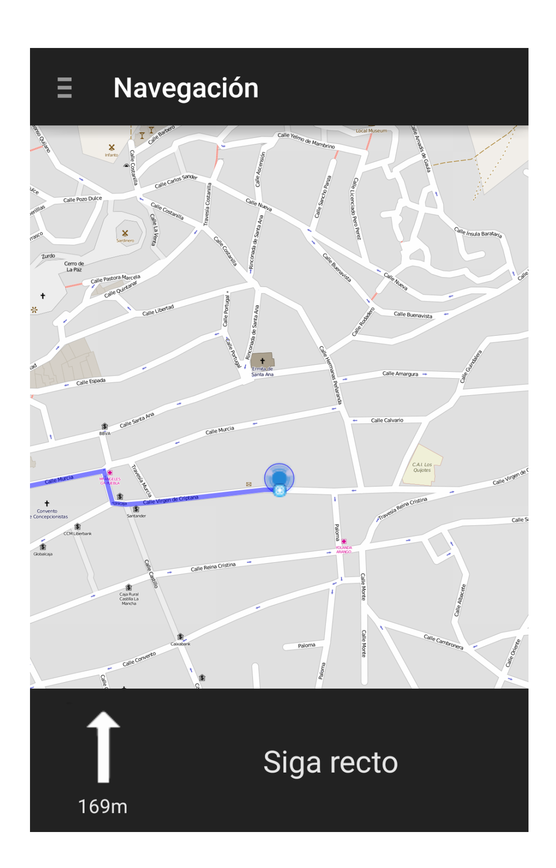


Figura 6.13: Inicio de ruta

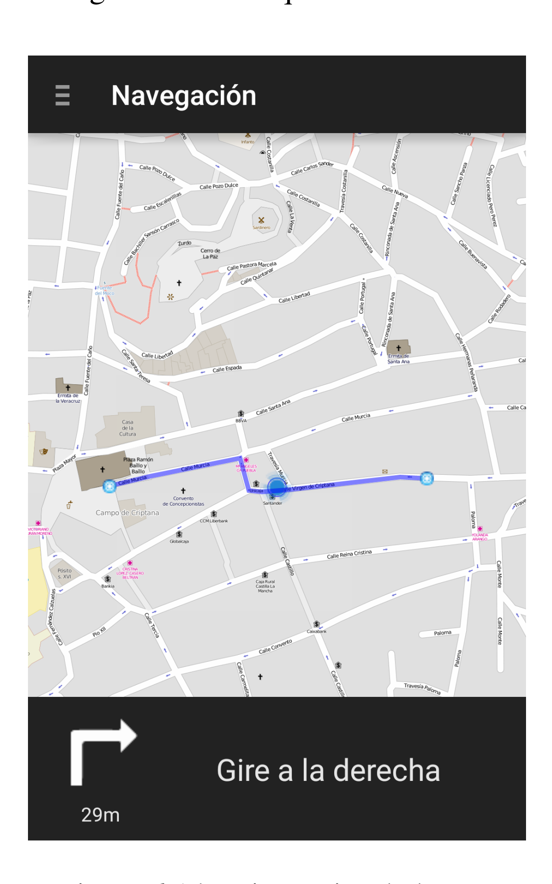


Figura 6.14: Primer giro de la ruta

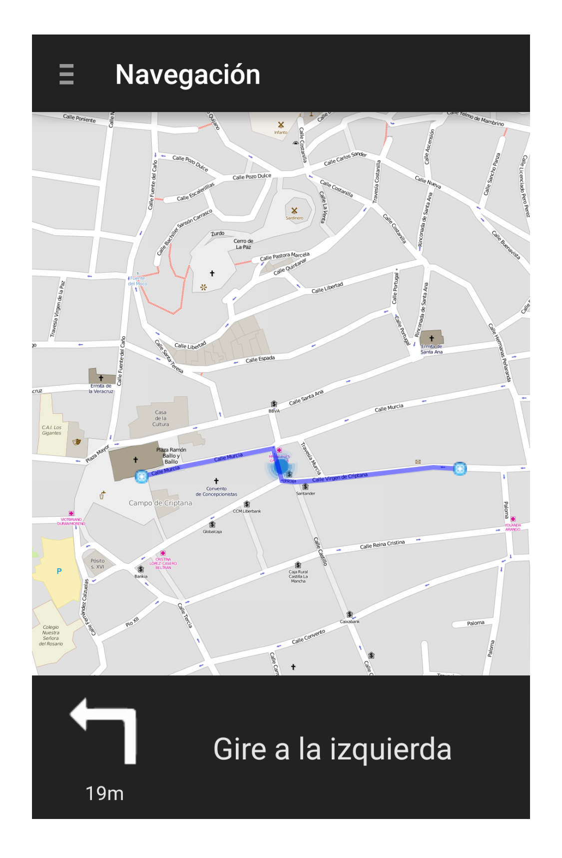


Figura 6.15: Segundo giro de la ruta

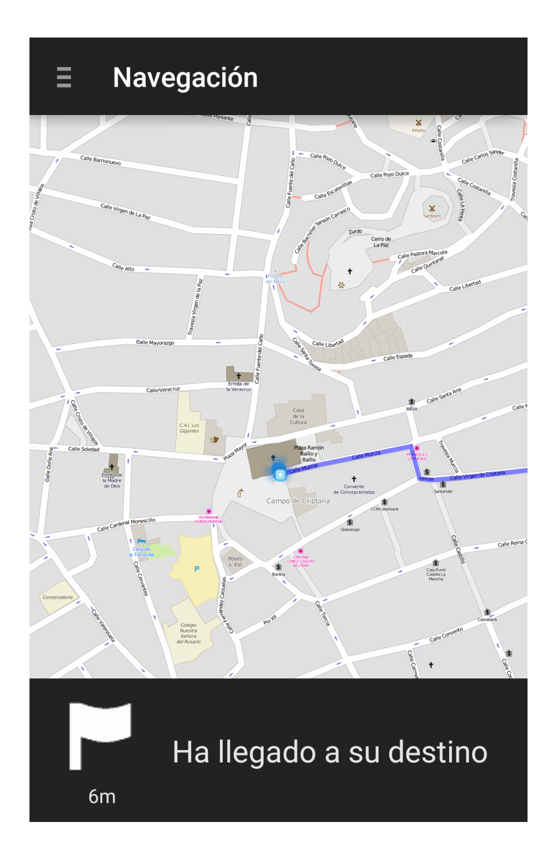


Figura 6.16: Llegada al destino, fin de la ruta

6.3 Repositorio

El código desarrollado durante el transcurso del proyecto y la documentación de esta memoria se encuentran disponibles en uno de los repositorios de *BitBucket*.

Para descargar todo el repositorio puede utilizar el siguiente comando:

```
git clone https://bitbucket.org/cr4mos/tfg-sgpcmii.git
```

Si sólo se desea descargar las aplicaciones finales, diríjase a:

```
https://bitbucket.org/cr4mos/tfg-sgpcmii/downloads
```

Aunque si se dispone de *Play Store* en el *smartphone* Android se puede descargar e instalar las aplicaciones más fácilmente accediendo a:

```
https://play.google.com/store/apps/details?id=es.uclm.esi.tfg.naviganto
https://play.google.com/store/apps/details?id=es.uclm.esi.tfg.navigantobluetooth
https://play.google.com/store/apps/details?id=es.uclm.esi.tfg.navigantoWear
```

Capítulo 7

Conclusiones

En el actual capítulo se presentan las conclusiones extraídas del desarrollo del TFG. Se comienza hablando de cómo se han alcanzado los diferentes objetivos, se continúa describiendo posibles usos del sistema en entornos diferentes a los propuestos inicialmente; y se termina enumerando posibles líneas de trabajo futuro que ampliarían la funcionalidad del presente proyecto.

7.1 Objetivos cumplidos

Tras finalizar el desarrollo del trabajo propuesto, se puede considerar que se han alcanzado satisfactoriamente los **objetivos específicos** planteados en el Capítulo 2:

- El **desarrollo del sistema de posicionamiento** se abordó en las iteraciones 1 y 2 (ver sección 5.3.1 y sección 5.3.2). Tras su implementación el sistema fue capaz de mostrar la posición del usuario representada en un mapa.
- El **desarrollo del sistema de navegación** se abordó en las iteraciones 4, 5, 6 y 7 (ver sección 5.3.4, sección 5.3.5, sección 5.3.6 y sección5.3.7). Tras su implementación el sistema fue capaz de navegar desde la posición actual de usuario hasta la localización deseada.
- El **desarrollo del sistema de reproducción de audio** se abordó en la iteración 8 (ver sección 5.3.8). Tras su implementación fue posible navegar por una ruta siguiendo las indicaciones sonoras del sistema.
- El **desarrollo del sistema de comunicación con otros dispositivos** se abordó en la iteración 9 (ver sección 5.3.9). Tras su implementación fue posible intercambiar mensajes entre los diferentes dispositivos del sistema.
- El **desarrollo del sistema vibratorio** se abordó en la iteración 9 (ver sección 5.3.9). Tras su implementación fue posible hacer vibrar todos los componentes del sistema a las frecuencias oportunas.

Por ello, se puede concluir que se ha culminado el **objetivo general** del proyecto: «desarrollo de un sistema de navegación por satélite que se comunique con el usuario por medio de la vibración de alguno de sus componentes».

7.2 Otros usos del sistema

Al tratarse de un trabajo dirigido a un público específico: peatones, ciclistas y motoristas; podría suponerse que son los únicos que podrían verse beneficiados del mismo, pero no es así.

Por un lado, los conductores de cualquier tipo de vehículo motorizado (coches, camiones, etc) podrían usar el sistema desarrollado en este trabajo para realizar sus rutas. *Naviganto* supone una alternativa real a las aplicaciones de navegación para *smpartphones* actuales siendo la única que puede comunicarse por vibraciones además de por la pantalla y el sonido.

Por otro lado, las personas que sufran algún tipo de discapacidad visual que provoque una disminución de su visión podrá utilizar *Naviganto* para sus desplazamientos, especialmente los discapacitados con la visión de largo alcance. Puesto que el sistema no se ha desarrollado específicamente para ellos, sólo podrían tener dificultad en la configuración inicial de los dispositivos vibratorios y en la selección del destino. Estas tareas pueden hacerlas terceras personas ya que sólo deben ejecutarse al comienzo de la ruta.

7.3 Líneas de trabajo futuro

A pesar de haber terminado el proyecto cumpliendo los objetivos propuestos inicialmente, hay aspectos del sistema que podrían ser pulidos para mejorarlo en general o para ampliar la cuota de usuarios finales del mismo. Estas futuras líneas de trabajo podrían ser:

Integración para discapacitados visuales
Tal y como se mencionó en la sección anterior, el sistema desarrollado en este proyecto podría usarse por discapacitados visuales para sus desplazamientos. Para que no sea necesaria una tercera persona que configure los vibradores y seleccione el destino de la ruta, se podría incluir un módulo de reconocimiento de voz que sustituya todas las posibles acciones que se realizan en *Naviganto*.

Implementación de funciones actuales sin conexión a Internet
A pesar de que a día de hoy es difícil encontrar personas que no dispongan de conexión a Internet en su *smartphone*, es posible que en algunos momentos no sea posible hacer uso de dicha conexión: ausencia de cobertura, viajes a otros países, etc. Para estas circunstancias sería interesante tener en la aplicación los mapas del lugar, algunas instrucciones sonoras y algunas rutas precalculadas.

Implementación de nuevas funcionalidades
Entre las nuevas funcionalidades que se podrían incluir en *Naviganto* podemos destacar dos que ya existen en otras aplicaciones de navegación para vehículos como:

- Representación visual del límite de velocidad de la vía en función del vehículo que se utilice para el desplazamiento e implementar una forma de avisar al usuario cuando sobrepase dicho límite.

- Representación visual del estado del tráfico de la vía por la que se circule.

Además, resultaría interesante integrar dentro del sistema una funcionalidad que no se ha podido encontrar en el resto de aplicaciones estudiadas en la sección 3.2.3: aviso de radares fijos para los desplazamientos con vehículos motorizados. Puesto que la información de la localización de los radares es pública [1] y es legal avisar de dichos radares en España [Bor14], resultaría una funcionalidad muy interesante y fácil de integrar con avisos vibratorios.

Desarrollo para otras plataformas

En la actualidad, sólo tendría sentido portar el sistema desarrollado sobre Android con un 81 % de cuota de mercado a su principal competidor: iOS con el 12,9 % de cuota de mercado.

Para que esta portabilidad tuviera éxito habría que modificar la forma en la que el sistema se comunica con los *wearables* (ya que en iOS no estarían disponibles los *Play Services*) u olvidarse de los *wearables* con Android Wear, pero no de los *wearables*. Aunque en la actualidad solamente los dispositivos con Android Wear permiten la manipulación a voluntad de su vibrador (ver sección 3.4.1), a lo largo de 2015 Apple presentará su propio *smartwatch* con su propio SO: Watch OS [2]. Como se espera que este SO sea el competidor de Android Wear, también se espera que permita manipular el vibrador del dispositivo.

[1] http://www.dgt.es/es/el-trafico/control-de-velocidad/
[2] https://www.apple.com/es/watch/overview/

ANEXOS

Anexo A

Conclusión personal

El desarrollo de este TFG pone fin a mis años como estudiante universitario. A lo largo de este tiempo, he adquirido numerosos conocimientos que me han servido para llevar a buen puerto este desarrollo. Sin embargo, también he tenido que enfrentarme a tecnologías y formas de programar que no conocía como es el desarrollo para Android y, especialmente, Android Wear. Esto me ha hecho descubrir mi capacidad de adaptación a nuevas tecnologías, ver la repercusión de realizar desarrollos eficientes en entornos con recursos reducidos y comprender la importancia de ejecutar en diferentes *threads* la *interfaz* gráfica y las peticiones a Internet.

Como pienso que el mercado de aplicaciones móviles Android y Android Wear se encuentra en plena expansión, considero que este proyecto me ha permitido conseguir conocimientos que me serán útiles en un futuro no muy lejano.

Por otro lado, con el desarrollo de este proyecto he podido comprobar que los desarrollos en Informática permiten ver resultados muy rápidamente a costes muy bajos o prácticamente nulos. Además, me ha permitido tomar en consideración lo aprendido durante la carrera para verificar que es sólo una ínfima parte de lo que hay fuera. Estos hechos, han reforzado mis ganas de continuar aprendiendo más sobre el ámbito de la Informática.

Por todo ello, considero que este proyecto se ha ajustado a lo que yo buscaba, y es el trabajo que más he disfrutado haciendo en toda la carrera.

Anexo B

Manual de usuario

El presente anexo se presenta un manual de usuario destinado a explicar todas las funcionalidades desarrolladas del sistema desarrollado en este proyecto.

B.1 Introducción

El sistema consta de tres aplicaciones diferentes:

- **Naviganto** Encargado de implementar todas las funciones de navegación por satélite y de comunicarse con los complementos vibratorios y activarlos.
- **Naviganto Bluetooth** Encargado de implementar el complemento vibratorio en Android.
- **Naviganto Wear** Encargado de implementar el complemento vibratorio en Android Wear.

Para el funcionamiento del sistema basta con instalar la aplicación *Naviganto* en nuestro dispositivo; pero sólo con esta aplicación no podremos hacer uso de la vibración para que nos guíe en nuestra navegación. Por tanto, para el funcionamiento completo del sistema necesitaremos dos dispositivos como mínimo: uno colocado en la parte izquierda de nuestro cuerpo y otro que colocaremos en la parte derecha. En el cuadro B.1 se resumen todas las posibles combinaciones válidas entre las aplicaciones del sistema para hacer uso del funcionamiento completo.

Naviganto	Naviganto Bluetooth	Naviganto Wear
1 (izquierda)	1 (derecha)	0
1 (derecha)	1 (izquierda)	0
1	2 (izquierda y derecha)	0
1 (izquierda)	0	1 (derecha)
1 (derecha)	0	1 (izquierda)
1	0	2 (izquierda y derecha)
1	1 (izquierda)	1 (derecha)
1	1 (derecha)	1 (izquierda)

Cuadro B.1: Resumen de combinaciones entre las aplicaciones del sistema

B.2 Instalación

En los siguientes apartados se exponen los requisitos mínimos para instalar cada una de las aplicaciones del sistema y los métodos de instalación.

B.2.1 Naviganto

Requisitos mínimos

Los requisitos mínimos que debe cumplir un *smartphone* para instalar y hacer funcionar *Naviganto* se describen a continuación:

- **Android** 2.3.3 Gingerbread o superior
- **2,9 MB** de espacio disponible como mínimo
- Hardware **GPS**
- Acceso a **Internet**

De igual forma, aunque no se consideran requisitos mínimos para la aplicación *Naviganto*, son necesarios los siguientes elementos para hacer uso del funcionamiento completo del sistema:

- Hardware **Bluetooth**
- Hardware **Vibrador**

Proceso de instalación

Para instalar automáticamente la aplicación desde el *Play Store* diríjase la dirección que le mostramos a continuación, pulse sobre el botón *Instalar* y acepte las condiciones que se le muestran:

```
https://play.google.com/store/apps/details?id=es.uclm.esi.tfg.
   naviganto
```

Si desea hacer una instalación manual, deberá activar los *orígenes desconocidos* de su Android [1] y descargar el *apk* del repositorio del proyecto:

```
https://bitbucket.org/cr4mos/tfg-sgpcmii/downloads/main-release.apk
```

B.2.2 Naviganto Bluetooth

Requisitos mínimos

Los requisitos mínimos que debe cumplir un *smartphone* para instalar y hacer funcionar *Naviganto Bluetooth* se describen a continuación:

[1] http://www.elandroidelibre.com/2013/06/como-instalar-aplicaciones-fuera-de-google-play-con-seguridad.html

- **Android** 2.3.3 Gingerbread o superior
- **911 KB** de espacio disponible como mínimo
- Hardware **Bluetooth**
- Hardware **Vibrador**

Proceso de instalación

Para instalar automáticamente la aplicación desde el *Play Store* diríjase la dirección que le mostramos a continuación, pulse sobre el botón *Instalar* y acepte las condiciones que se le muestran:

```
https://play.google.com/store/apps/details?id=es.uclm.esi.tfg.
    navigantobluetooth
```

Si desea hacer una instalación manual, deberá activar los *orígenes desconocidos* de su Android (al igual que para *Naviganto*) y descargar el *apk* del repositorio del proyecto:

```
https://bitbucket.org/cr4mos/tfg-sgpcmii/downloads/bluetooth-release.
    apk
```

B.2.3 Naviganto Wear

Requisitos mínimos

Los requisitos mínimos que debe cumplir un *wearable* para instalar y hacer funcionar *Naviganto Wear* se describen a continuación:

- **Android Wear** 1.0 o superior
- **1,8 MB** de espacio disponible como mínimo
- Hardware **Bluetooth**
- Hardware **Vibrador**
- **Conexión** con un *smartphone* Android 4.4.2 KitKat o superior

Proceso de instalación

Para instalar automáticamente la aplicación desde el *Play Store* diríjase la dirección que le mostramos a continuación, pulse sobre el botón *Instalar* y acepte las condiciones que se le muestran:

```
https://play.google.com/store/apps/details?id=es.uclm.esi.tfg.
    navigantoWear
```

Si desea hacer una instalación manual, puede utilizar la herramienta *Android Wear APK Tools* [2] y descargar el *apk* del repositorio del proyecto:

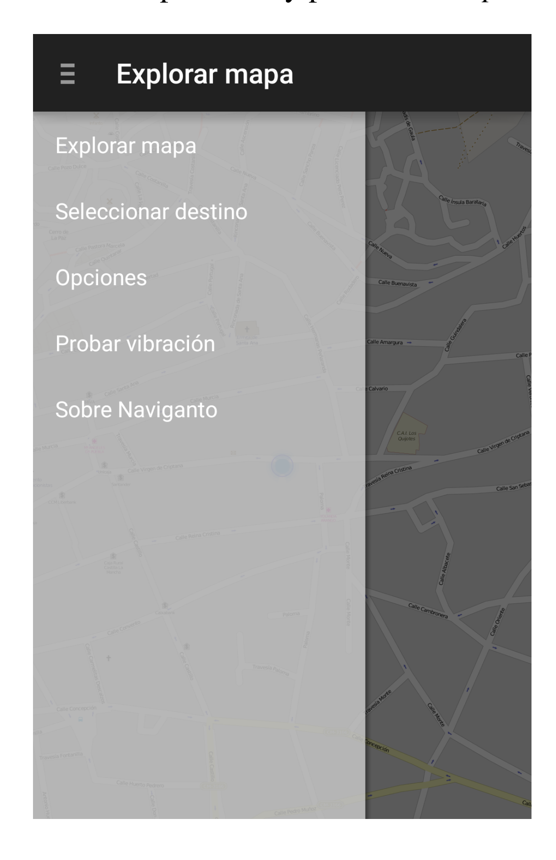
```
https://bitbucket.org/cr4mos/tfg-sgpcmii/downloads/wear-release.apk
```

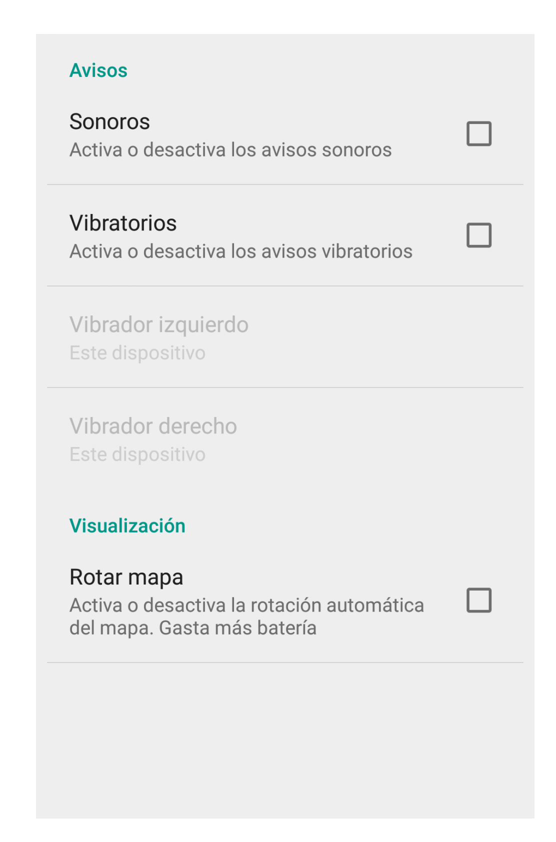
B.3 Uso de Naviganto

En los siguientes apartados se explican detalladamente las opciones implementadas en *Naviganto*, cómo configurarlas y los dos casos de uso de la aplicación.

B.3.1 Configuración

Para acceder a las diferentes opciones de configuración de *Naviganto* acceda a la barra lateral de la aplicación y pulse sobre *Opciones*:





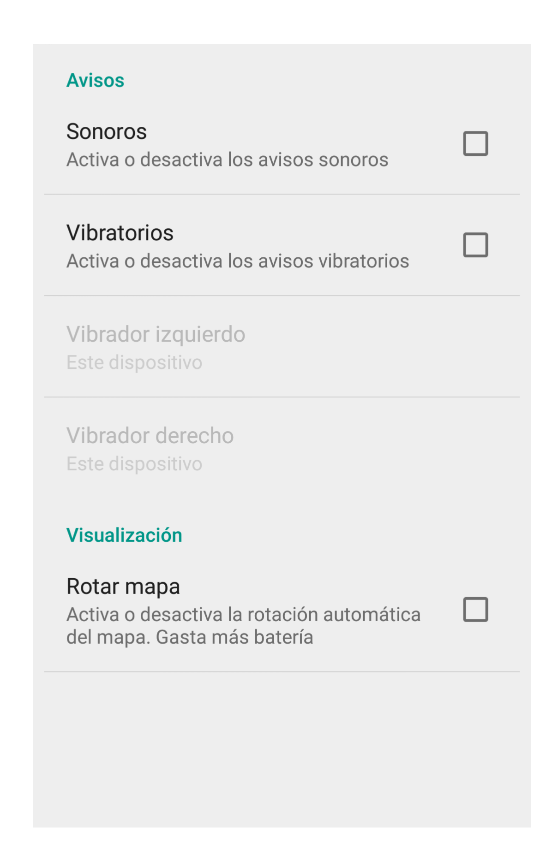
Aquí podrá configurar las opciones descritas en los siguientes apartados.

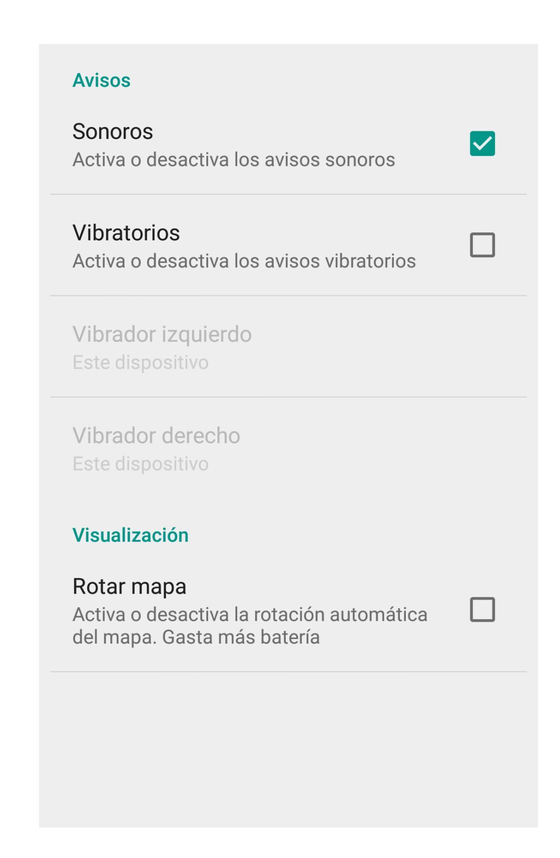
Avisos sonoros

Los avisos sonoros hacen referencia a la posibilidad de escuchar cada una de las posibles instrucciones que tengamos que realizar.

Para activar o desactivar dichos avisos basta con marcar o desmarcar la casilla del menú:

[2] http://forum.xda-developers.com/smartwatch/other-smartwatches/tool-android-wear-apk-tools-sideload-t2929177





Esta configuración se almacena en el *smartphone* de forma que al cerrar y volver a abrir la aplicación se mantenga presente.

Avisos vibratorios

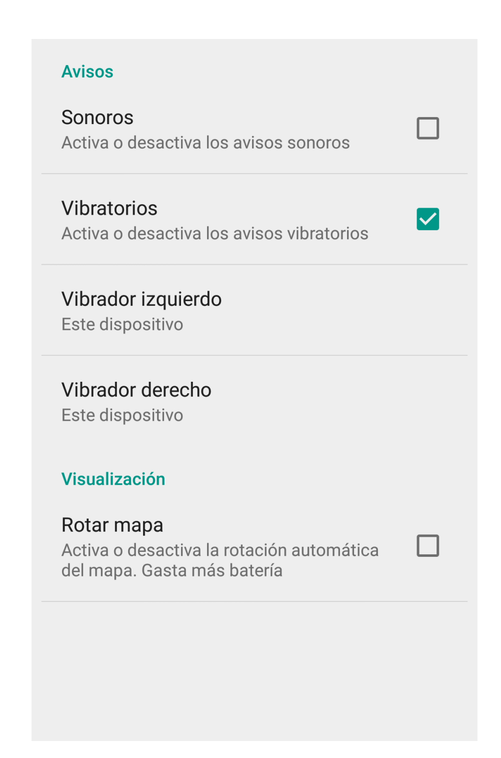
Los avisos vibratorios hacen referencia a la posibilidad de que el *smartphone*, junto con otro complemento vibratorio, nos avisen de las instrucciones que tengamos que realizar.

Para activar o desactivar dichos avisos hay que activar la casilla del menú y después seleccionar qué dispositivo colocaremos en la parte izquierda de nuestro cuerpo y cuál en la parte derecha.

Para que aparezcan dichos dispositivos en la lista debemos tener:

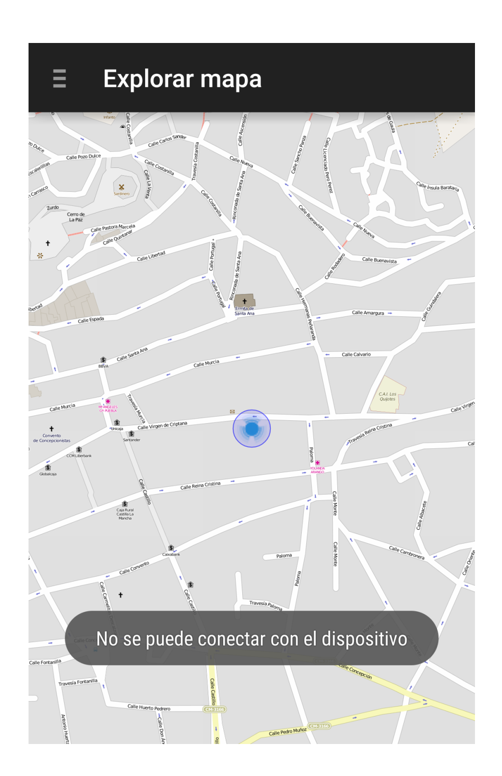
- En caso de utilizar **Naviganto Bluetooth** como complemento, pareados los dos dispositivos por medio de *bluetooth*.
- En caso de utilizar **Naviganto Wear** como complemento, sincronizado nuestro *wearable* con el *smartphone* por medio de los *Play Services*.

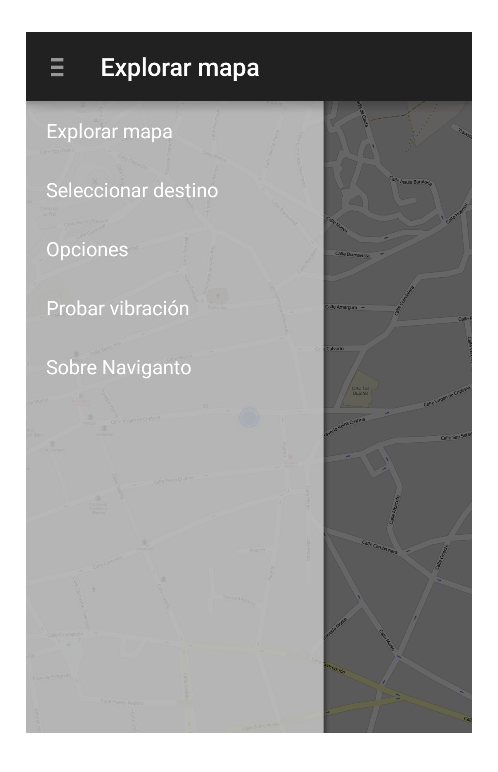
Una vez seleccionados los dispositivos, cuando pulsemos sobre la tecla de retroceso se procederá a la conexión con dichos dispositivos. En este punto deberán estar activas las aplicaciones en los complementos que dispongan de *Naviganto Bluetooth* y es opcional que se encuentre ejecutándose la aplicación *Naviganto Wear* en los *wearables*.





En caso de que la conexión no tenga éxito, *Naviganto* nos avisará con un mensaje; en cambio, si la conexión tiene éxito, *Naviganto* nos permitirá realizar una prueba de conexión desde el menú de la barra lateral pulsando sobre *Probar vibración*.



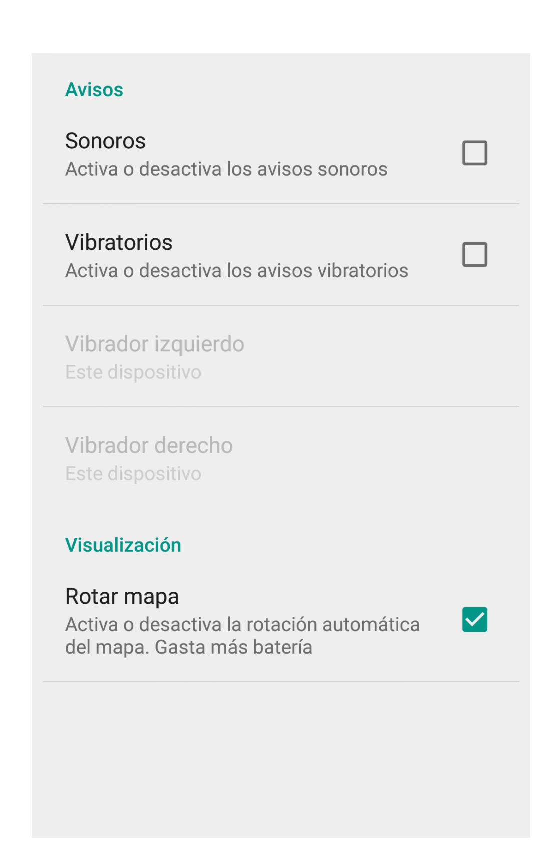


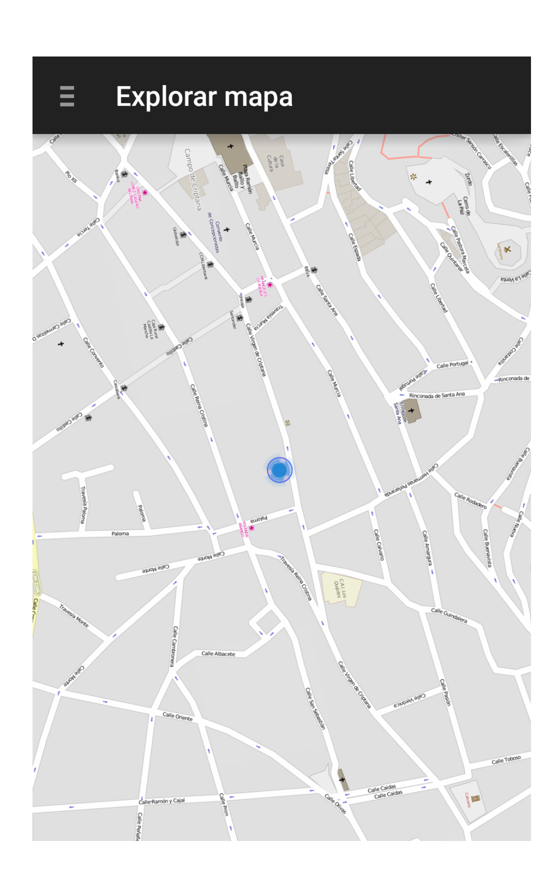
En este momento podríamos iniciar una ruta y nos guiaríamos con las vibraciones de los dispositivos, aunque si reiniciásemos la aplicación tendríamos que volver a configurar la vibración.

Rotación de mapa

La rotación de mapa hace referencia a la posibilidad de orientar el mapa mostrado en función de la orientación actual del *smartphone*. Esta rotación conlleva un aumento considerable de uso del procesador y, por tanto, el aumento del consumo de batería.

Para activar o desactivar esta opción basta con marcar o desmarcar la casilla del menú:





Esta configuración se almacena en el *smartphone* de forma que al cerrar y volver a abrir la aplicación se mantenga presente.

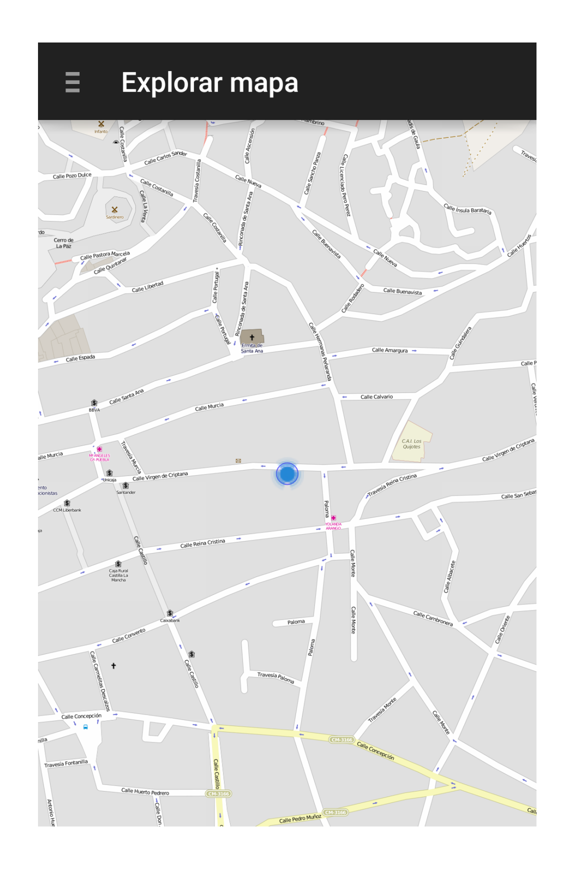
B.3.2 Navegación

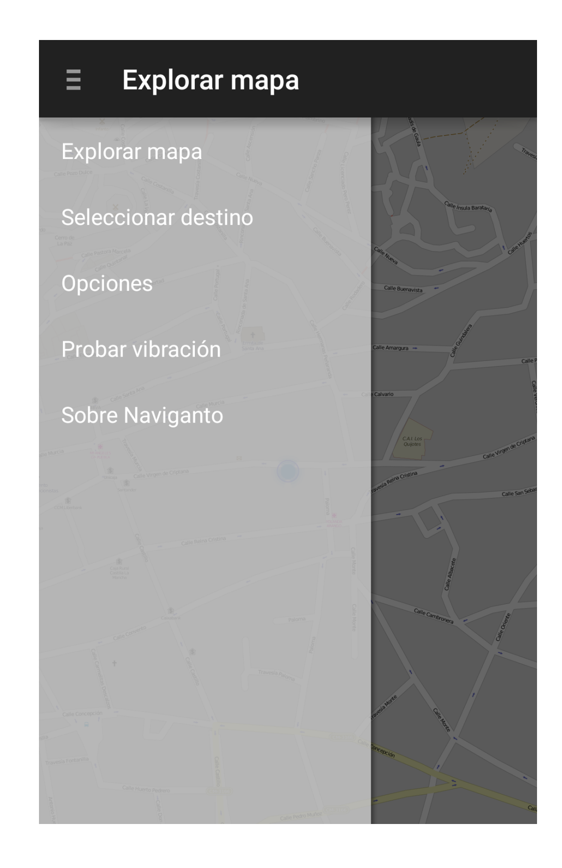
Explorar mapa

El caso de uso más simple de *Naviganto* es el de *explorar mapa* y, para empezar a explorarlo, basta con abrir la aplicación y nos encontraremos haciéndolo. En caso de que nos encontremos siguiendo una ruta y decidamos abandonarla para comenzar a explorar el mapa sólo debemos seleccionar esta opción en el menú de la barra lateral.

Explorar mapa permite inspeccionar el mapa deslizándolo en cualquier dirección y aumentar o disminuir el *zoom* utilizando dos dedos (como en las fotos).

Para *explorar mapa* sólo es necesario disponer de conexión a Internet, aunque se recomienda tener activado el GPS para que se muestre nuestra posición actual centrada en el mapa. En caso de tener desactivado el GPS nos centrará la última posición registrada.

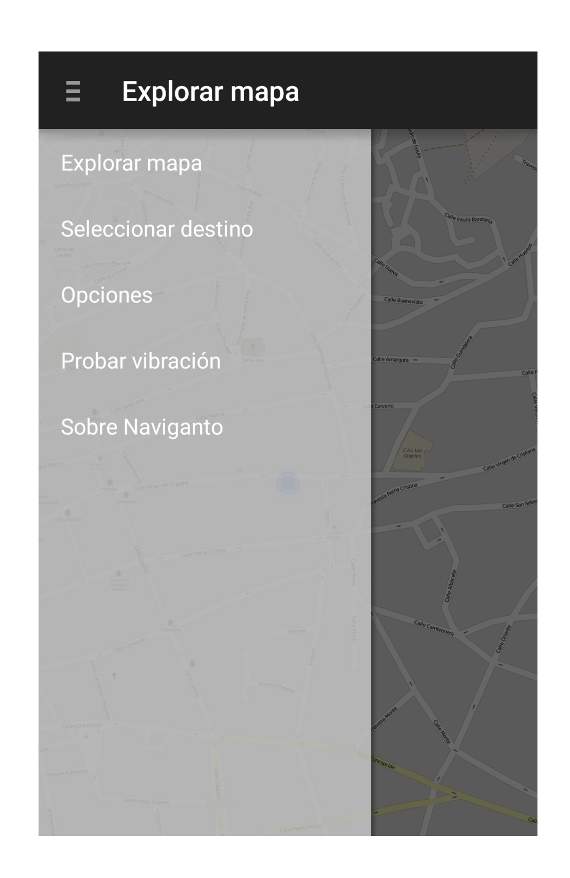


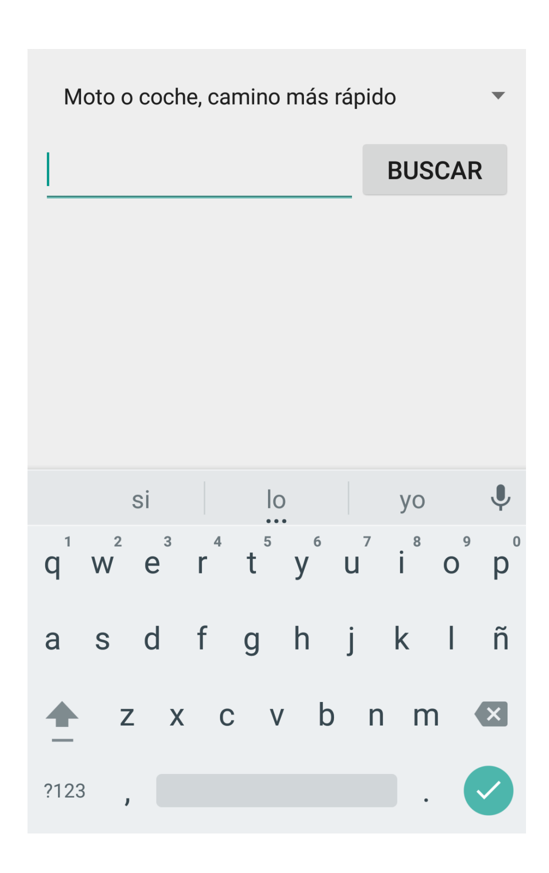


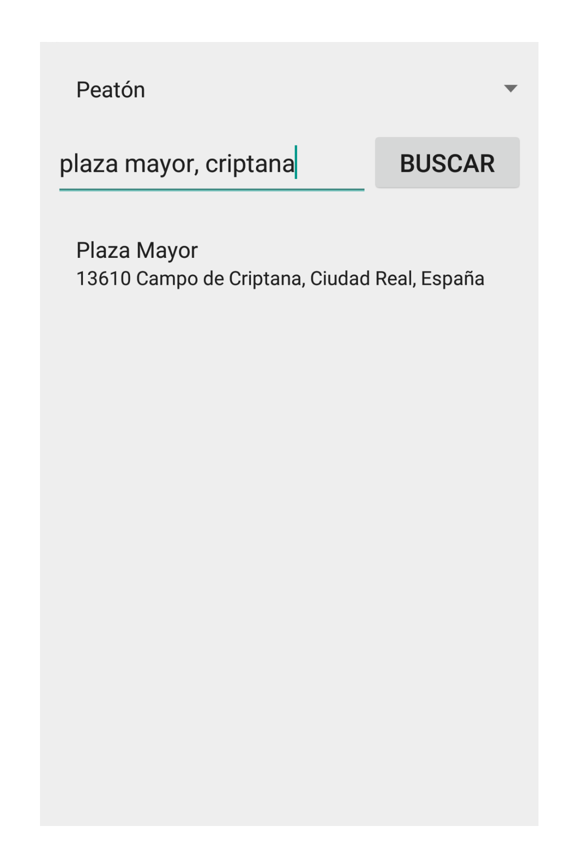
Guiado por ruta

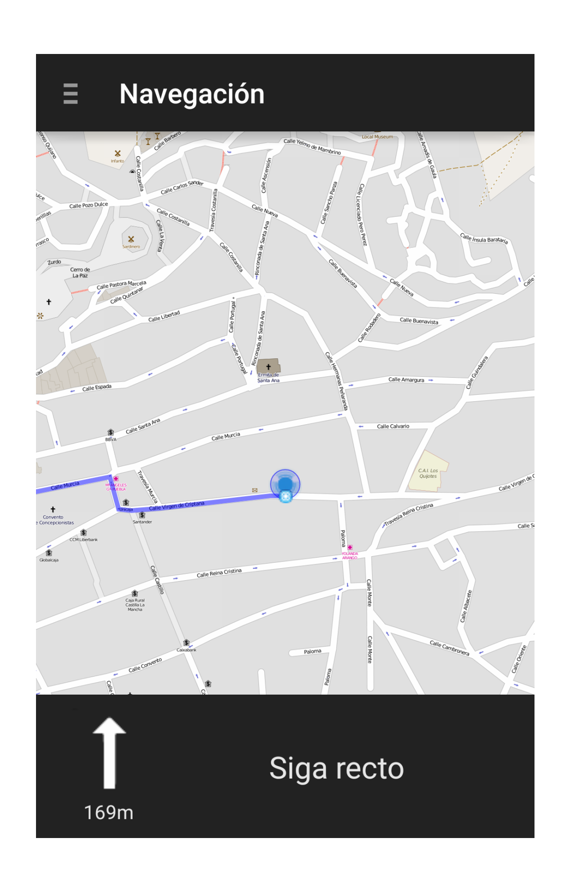
Para que *Naviganto* nos *guíe por una ruta* debemos seleccionar el destino al que queremos desplazarnos y el método que utilizaremos: moto, bici o andando. Para especificar dichos datos accedemos al menú lateral y pinchamos sobre *Seleccionar destino*. Aquí introducimos la dirección final deseada, el método de desplazamiento y pinchamos sobre buscar para que se nos muestren las posibles coincidencias. Llegados a este punto, sólo resta seleccionar el destino que queramos y comenzar el viaje siguiendo las instrucciones hasta llegar al lugar especificado.

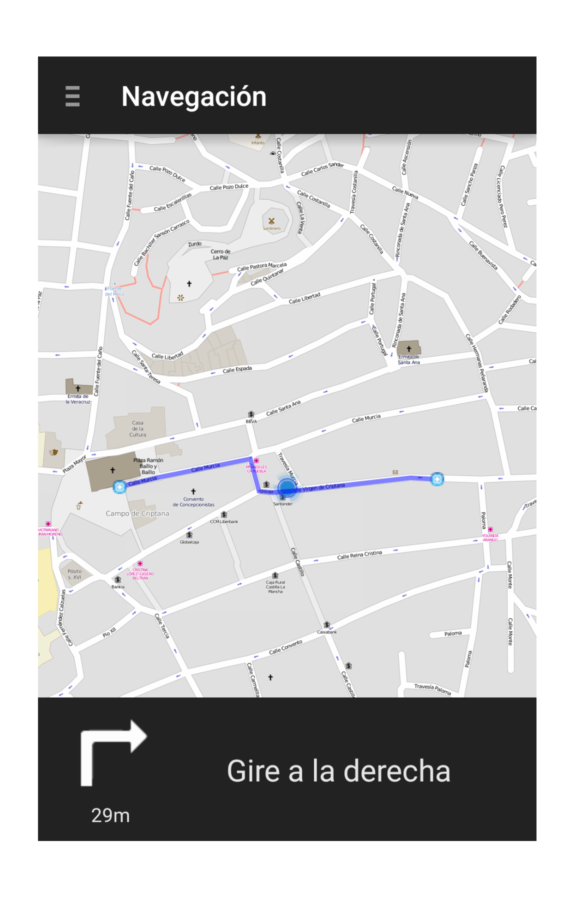
Para la búsqueda y el cálculo de la navegación es necesario disponer de conexión a Internet; y para la navegación por la ruta solamente es necesario tener activado el GPS. De todos modos, se recomienda disponer de conexión a Internet durante todo el desplazamiento por la ruta para poder activar los avisos sonoros y para recalcular la ruta en el caso de que nos desviemos de la misma.

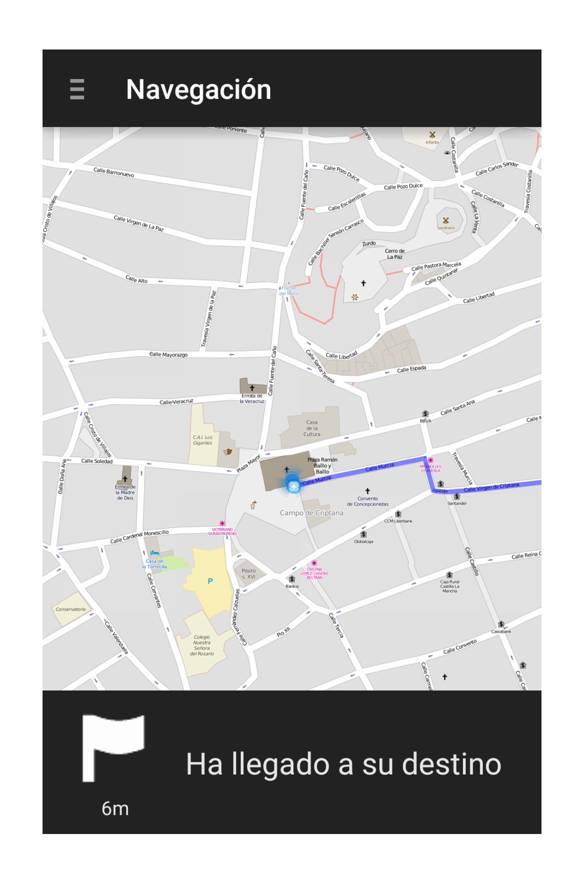

Explorar mapa
Explorar mapa
Seleccionar destino
Opciones
Probar vibración
Sobre Naviganto


Moto o coche, camino más rápido
BUSCAR


Peatón
plaza mayor, criptana
BUSCAR
Plaza Mayor
13610 Campo de Criptana, Ciudad Real, España


Navegación
Siga recto
169m


Navegación
Gire a la derecha
29m


Navegación
Ha llegado a su destino
6m

Anexo C

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that

work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors,

SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover

Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the

publisher.

- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your

option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.
5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

5. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this

License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular

copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

10. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Referencias

[Boe12] Eduardo Boemo. Desarrollo de Sistema de guiado GPS para invidentes sobre un teléfono inteligente. 2012.

[Bor14] J. Arias Borque. Avisadores, detectores e inhibidores de radar. 2014.

[Coa14a] Steve Coast. Licencia de los mapas de OSM. https://en.wikipedia.org/wiki/Open_Database_License, Diciembre 2014.

[Coa14b] Steve Coast. Politica de uso de Nominatim. http://wiki.openstreetmap.org/wiki/Nominatim_usage_policy, Diciembre 2014.

[dFCdC03] Universidad de Florida Circuit de Catalunya. Incidencia del uso del teléfono móvil y los aparatos telemáticos en la atención del conductor. 2003.

[Eng14] Engadget. Estas zapatillas inteligentes marcarán la ruta de tus carreras con vibraciones. http://es.engadget.com/2014/07/24/lechal-zapatillas-inteligentes/, Julio 2014.

[Foo15] Footlocker. Nike+ Training. http://www.footlocker.com/promotion/promoId:5003172/, Enero 2015.

[Goo14a] Google. API Adroid Wear. https://developer.android.com/training/wearables/apps/index.html, Diciembre 2014.

[Goo14b] Google. Licencias de las API de Google. https://developers.google.com/maps/licensing, Diciembre 2014.

[Lit13] Steve Litchfield. Support until 2016 confirmed by Nokia's Weber. 2013.

[Mat13] Tomás Merino Mateo. Desarrollo en iOS de aplicaciones de guiado GPS para discapacitados visuales. 2013.

[Mer13] Lothar Merk. *Pervasive Computing*. Springer, 2013.

[Mic14] Microsoft. Licencias de las API de Microsoft. http://www.microsoft.com/maps/Licensing/licensing.aspx, Diciembre 2014.

[Nok14] Nokia. Licencias de las API de HERE. https://developer.here.com/get-started#/10134035, Diciembre 2014.

[Par13] Sergio Parra. ¿Cómo era el primer GPS de la historia? 2013.

[Pre10] Roger S. Pressman. *Ingenieria del Software - Un Enfoque Practico*. McGraw-Hill Companies, 2010.

[Pul14] Allianz Risk Pulse. Seguridad vial y distracciones del conductor. 2014.

[RL13] Michael Shirer Ramon Llamas, Ryan Reith. Android Pushes Past 80 % Market Share While Windows Phone Shipments Leap 156.0 % Year Over Year in the Third Quarter, According to IDC. 2013.

[Ser14] Jorge Serrano. Las distracciones más frecuentes al volante. 2014.

[Som05] Ian Sommerville. *Ingeniería del software*. Pearson Educacion, 2005.

[TTH92] Ronald Baecker Thomas T. Hewett. *SIGCHI Curricula for Human-Computer Interaction*. ACM, 1992.

[Val12] Josefa Valcárcel. Las principales cifras de la Siniestralidad Vial. 2012.

[Wik14a] Wikipedia. Satellite navigation. https://en.wikipedia.org/wiki/Satellite_navigation#History_and_theory, Diciembre 2014.

[Wik14b] Wikipedia. Sistemas de Posicionamiento por Satélites actuales. https://es.wikipedia.org/wiki/Sistema_global_de_navegaci%C3%B3n_por_sat%C3%A9lite, Diciembre 2014.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **esi-tfg** que se puede encontrar en:
https://bitbucket.org/arco_group/esi-tfg